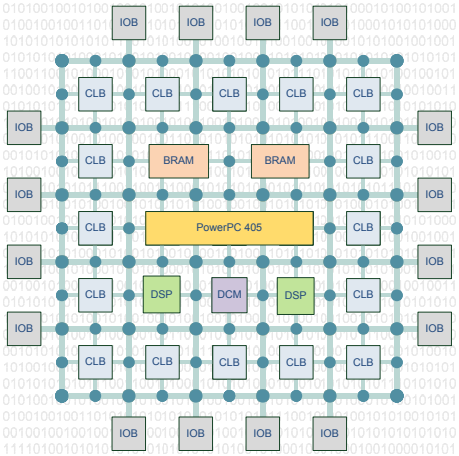
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2008

Nelson José Valente da Silva

Executivo de Tempo-real para Processadores Embutidos em FPGA




Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática
2008

Nelson José Valente da Silva

Executivo de Tempo-real para Processadores Embutidos em FPGA

Dissertação apresentada à Universidade de Aveiro, para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações (M.I.E.E.T.), realizada sob a orientação científica do Prof. Doutor Arnaldo Oliveira e do Prof. Doutor Luís Almeida, Professores do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

presidente

Doutor Valeri Skliarov
Professor Catedrático da Universidade de Aveiro

vogais

Doutor Mário João Barata Calha
Professor Auxiliar do Departamento de Informática da Faculdade de Ciências da Universidade de Lisboa

Doutor Luís Miguel Pinho de Almeida
Professor Auxiliar da Universidade de Aveiro

Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar Convidado da Universidade de Aveiro

agradecimentos

É com muito gosto que aproveito esta oportunidade para agradecer a todos os que me ajudaram. Começo por agradecer aos meus orientadores, o Prof. Doutor Arnaldo Oliveira e o Prof. Doutor Luís Almeida, por terem-me proposto este trabalho e disponibilizado os meios para a sua realização, pela orientação e apoio prestados, sempre com elevada disponibilidade e acompanhamento.

Gostaria ainda de agradecer aos meus amigos e colegas do laboratório, em particular ao Domingos Terra, ao Samuel Madail, ao João Puga e ao Sérgio Tafula, pela constante demonstração de camaradagem e entreajuda.

Por último, mas de uma maneira muito especial, agradeço à minha namorada Sandra, aos meus pais Maria e José e à minha irmã Patrícia, por todo o apoio e força que me deram, ao longo da minha formatura e em particular na conclusão deste mestrado. A eles dedico este trabalho.

Palavras-chave

Processadores Embutidos, Sistemas Embutidos, Sistemas Integrados, Sistemas de Tempo-real, Executivos de Tempo-real.

Resumo

Devido à grande evolução da tecnologia microelectrónica ao longo das últimas décadas, actualmente é possível a construção de circuitos integrados extremamente complexos, contendo diversas centenas de milhões de transístores. Este facto tornou o conceito de sistema integrado uma realidade. Uma das consequências da evolução da tecnologia foi a disponibilização de dispositivos lógicos complexos reconfiguráveis como as *FPGAs*. Tais dispositivos permitem o desenvolvimento de sistemas integrados, com a vantagem de possibilitarem uma rápida fase de prototipagem.

A crescente capacidade computacional e a redução do tamanho levaram a que os microprocessadores passassem a estar embutidos em muitos dos equipamentos e aplicações do quotidiano. Dependendo da aplicação que o sistema embutido irá desempenhar, pode existir a necessidade da sua execução em intervalos de tempo bem definidos, e um atraso temporal na resposta a um determinado estímulo pode significar o colapso no controlo de todo um processo. Tais sistemas possuem restrições de tempo-real e requerem abordagens de concepção adequadas para o seu correcto funcionamento.

A grande motivação para a realização deste trabalho consistiu na possibilidade de integrar num único dispositivo físico reconfigurável, todos os componentes de uma arquitectura computacional especializada para sistemas de tempo-real, ou seja, integrar processador, memória, periféricos, executivo de tempo-real e aplicação, num único dispositivo físico.

Nesta dissertação são discutidas ideias, apresentadas arquitecturas e avaliadas implementações de um executivo de tempo-real para processadores embutidos em *FPGA*. O principal objectivo deste trabalho consiste em integrar e avaliar um executivo de tempo-real embutido em *FPGA*.

O ponto de partida deste trabalho foi a construção de um sistema integrado em *FPGA*, recorrendo às ferramentas de desenvolvimento *Xilinx EDK* (*Embedded Development Kit*). O sistema desenvolvido, designado por *MB-SoC* (*MicroBlaze - System-on-Chip*), reúne um conjunto de periféricos, dispositivos de entrada/saída, memória e o processador sintetizável *MicroBlaze* implementado em blocos lógicos configuráveis da *FPGA*. Posteriormente foi ainda desenvolvido um segundo sistema, designado por *PPC-SoC* (*PowerPC 405 - SoC*). Este é idêntico ao anterior, mas utiliza um processador de alto desempenho, implementado em lógica dedicada e interno à *FPGA*.

O executivo de tempo-real *OReK* (*Object-oriented Real-time Kernel*) foi portado, no âmbito deste trabalho, para os sistemas integrados desenvolvidos, *MB-SoC* e *PPC-SoC*.

Os resultados da avaliação efectuada permitem analisar as vantagens e as desvantagens, associadas às diversas implementações.

Keywords Embedded Processors, Embedded Systems, Systems-on-Chip, Real-time Systems, Real-time Kernels.

Abstract

Due to the great development of microelectronics technology over recent decades, today it is possible the construction of extremely complex integrated circuits, containing several hundred million transistors. This made the System-on-Chip concept a reality. One of the consequences of the evolution of technology was the provision of complex logic devices such as reconfigurable FPGAs. Such devices allow the System-on-Chip development with the advantage of allowing a rapid prototyping phase.

The increasing computational capacity and the reduction of the size led the microprocessors to be embedded in many of the equipment and applications in everyday life. Depending on the application that the embedded system will do, there may be a need for execution at well defined intervals of time, and a time delay in responding to a stimulus can mean the collapse in control of an entire process. Such systems have real-time restrictions and require appropriate design approaches for the proper functioning.

The major motivation for this work was the possibility of integrating into a single physical reconfigurable device, all components of an computational architecture specialized for real-time systems.

In this dissertation ideas are discussed, presented architectures and evaluated implementations of a real-time executive for embedded processors in FPGA. The main objective of this work is to integrate and evaluate an executive of real-time, embedded in FPGA.

The starting point of this work was the construction of an FPGA System-on-Chip, using the development tools named Xilinx EDK (Embedded Development Kit). The developed system, named *MB-SoC* (MicroBlaze - System-on-Chip), gathers a set of peripherals, input/output devices, memory and the soft-core MicroBlaze processor implemented in FPGA configurable logic blocks. Later was developed a second system, named *PPC-SoC* (PowerPC 405 - SoC). This is identical to the previous, but uses an high performance processor, implemented in dedicated logic and internal to the FPGA.

The real-time executive *OReK* (Object-oriented Real-time Kernel) was ported, within this work, to the developed integrated systems, *MB-SoC* and *PPC-SoC*.

The results of the evaluation can analise the advantages and the disadvantages, associated to the various implementations.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Sumário

Este capítulo apresenta o enquadramento do trabalho apresentado nesta dissertação. Primeiramente são introduzidos os conceitos de sistemas embutidos de tempo-real e de sistemas integrados reconfiguráveis. De seguida é apresentada a motivação para a realização deste trabalho e enumerados os respectivos objectivos. Por fim, é resumida a estrutura desta dissertação.

1.1 Sistemas Embutidos de Tempo-real

O aumento da capacidade computacional e a redução do tamanho levaram a que os microprocessadores passassem a estar presentes (ou embutidos) em muitos dos equipamentos e aplicações do quotidiano, tais como as telecomunicações, os transportes, os sistemas de segurança, a automação industrial, entre outras. Estes podem possuir complexidades tão díspares quanto sistemas com um único microcontrolador a sistemas com dezenas ou centenas de microcontroladores, interligados por redes de comunicação e dispondo de inúmeros periféricos.

A designação de sistema embutido provém do inglês *embedded system*, cuja definição pode ser: Um sistema computacional especializado, concebido para a realização de um conjunto predefinido de tarefas, muitas vezes de controlo, podendo estar inserido em sistemas complexos heterogéneos, isto é, sistemas que, para além da parte computacional, podem incluir também elementos eléctricos, pneumáticos, mecânicos, hidráulicos, ou outros.

Por este motivo, um sistema embutido é em regra geral desenvolvido com um fim específico e portanto, optimizado para as funcionalidades que irá desempenhar. É comum os sistemas embutidos possuírem restrições mais apertadas que os sistemas computacionais de uso geral ao nível do consumo de potência, área ocupada, requisitos temporais e custo.

Os sistemas embutidos tipicamente combinam componentes de *hardware* com aplicações de *software*, tendo em vista o aproveitamento das vantagens de ambos: O *hardware* possibilita rapidez e o *software* proporciona as facilidades da programação. Assim sendo, possuem a quantidade mínima de recursos de *hardware* necessários para o funcionamento do sistema, o que permite a utilização de dispositivos genéricos com funcionalidades programáveis, como os microcontroladores, e por outro lado, o *software* simplifica a implementação das funciona-

lidades incumbidas ao sistema embutido, o que contribui para uma diminuição temporal das fases de desenvolvimento e teste do sistema, e em última análise, para redução do custo global.

Dependendo das funcionalidades que um sistema embutido irá desempenhar, pode existir a necessidade deste actuar em intervalos de tempo bem definidos. Um atraso temporal na resposta a um determinado estímulo pode significar o colapso no controlo de todo um processo. Tais sistemas possuem restrições de tempo-real e requerem a execução de programas adequados para o seu funcionamento. Por sua vez, os programas podem ser baseados em técnicas de programação empíricas e de baixo nível, por exemplo, o uso da linguagem *assembly* e a construção de *device drivers* específicos, porém a complexidade crescente dos sistemas e a necessidade de reduzir o seu tempo de projecto, tem levado a uma utilização crescente de executivos e sistemas operativos multi-tarefa, os quais implementam camadas de abstracção do *hardware* e disponibilizam um conjunto de serviços que reduzem o tempo de desenvolvimento. No entanto, estas camadas intermédias de *software* consomem tempo de processamento e são também elas próprias por vezes uma fonte de não determinismo. O executivo é uma entidade de *software* responsável pela execução concorrente de processos ou tarefas e no caso dos sistemas de tempo-real, é ainda responsável por fazer cumprir ou verificar o cumprimento das restrições temporais que asseguram o correcto funcionamento do sistema.

1.2 Sistemas Integrados Reconfiguráveis

A grande evolução da tecnologia microelectrónica ao longo das últimas décadas permite actualmente a construção de circuitos integrados extremamente complexos, contendo diversas centenas de milhões de transístores. Este facto tornou o conceito de sistema integrado uma realidade.

Os sistemas integrados num único dispositivo físico (*chip*) é comum serem designados por *SoC* (*System-on-Chip*). Estes sistemas integram unidades de processamento, memória e periféricos específicos da aplicação, o que pode resultar em diversos ganhos ao nível do desempenho, custo, consumo de potência e área ocupada. Uma das consequências da evolução da tecnologia foi a disponibilização de dispositivos lógicos complexos reconfiguráveis como as *FPGAs*. Tais dispositivos permitem o desenvolvimento de sistemas integrados, com a vantagem de possibilitarem uma rápida fase de prototipagem.

São várias as definições existentes para a designação de sistemas reconfiguráveis, porém estes podem ser resumidos a sistemas cujas funcionalidades lógicas podem ser configuradas sem necessidade de alterar o dispositivo físico. Os sistemas reconfiguráveis têm ainda a particularidade do seu funcionamento interno ser definido após o processo de fabrico.

Dos dispositivos reconfiguráveis existentes, os mais relevantes para este trabalho são as *FPGAs* (*Field Programmable Gate Arrays*). Uma *FPGA* pode ser descrita como uma matriz de blocos programáveis (*CLBs - Configurable Logic Blocks*), envolta por blocos de entrada/saída (*IOBs - Input Output Blocks*) e conectados por recursos de interligação também configuráveis. As *FPGAs* possibilitam o projecto rápido e flexível de sistemas digitais complexos, integrados num único dispositivo. A figura 1.1 ilustra um exemplo da organização interna de uma *FPGA*.

A implementação de um circuito lógico numa *FPGA* é feita através da distribuição da lógica entre os blocos individuais programáveis, interligados com recursos de ligação configuráveis. É de salientar que os atrasos resultantes são grandemente influenciados pela distribuição da lógica e pela estrutura de encaminhamento. Assim sendo, o desempenho de circuitos mapeados em *FPGA* depende muito da eficácia das ferramentas *CAD* (*Computer Aided Design*), utilizadas na sua síntese e implementação.

A evolução das *FPGAs* traduziu-se em aumentos de desempenho, densidade, capacidade de processamento e diminuição do consumo de energia [Xil08d]. Actualmente estão disponíveis *FPGAs* com capacidade lógica equivalente a milhões de portas lógicas, na forma de recursos quer programáveis quer fixos, tais como *LUTs*, memórias, macro-células do tipo *DSP*, processadores de uso geral (ex. *PowerPC*), controladores para protocolos de comunicação (ex. *Ethernet*, *PCI Express*) e blocos de entrada/saída muito versáteis.

De referir ainda que cada fabricante implementa *FPGAs* com diferentes funcionalidades, arquitecturas e ferramentas de suporte. Neste trabalho foi utilizada uma *FPGA* da família *Virtex-II Pro* da *Xilinx* [Xil08n], porém o trabalho realizado é facilmente portável para outras *FPGAs*.

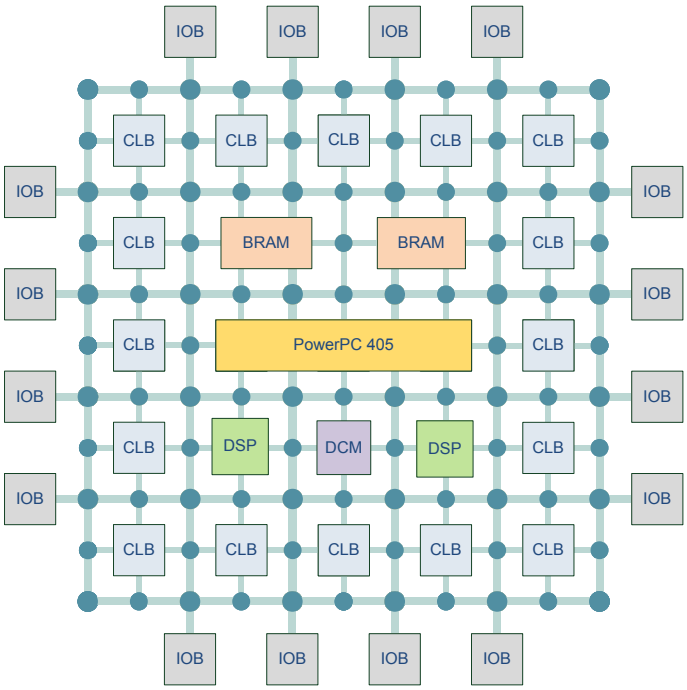


Figura 1.1: Estrutura interna de uma *FPGA*.

O trabalho descrito nesta dissertação utiliza sistemas que integram num único dispositivo físico diversos periféricos e o processador *MicroBlaze* [Xil08e] ou o *PowerPC 405* [Xil08i], sendo designados por *MB-SoC* e *PPC-SoC*, respectivamente.

1.3 Motivação

A principal motivação para a realização deste trabalho consistiu na possibilidade de integrar num único dispositivo físico, todos os componentes de uma arquitectura computacional especializada para sistemas de tempo-real. Tal sistema integrado deverá possuir um conjunto de recursos de *hardware* (processador, memória, periféricos, etc.) e de *software* (executivo e aplicação de tempo-real) que permitam a sua utilização num sistema de tempo-real. Para o efeito foi necessário proceder à escolha de um executivo de tempo-real e de *hardware* dedicado.

A motivação final para a concepção deste trabalho está relacionada com os conceitos e metodologias utilizadas, nomeadamente o uso de sistemas de tempo-real em processadores embutidos e ainda a utilização de *FPGAs* para implementar sistemas integrados. Existe ainda interesse no estabelecimento de um fluxo de projecto.

A eleição do executivo de tempo-real *OReK* (*Object-oriented Real-time Kernel*) foi motivada pelo suporte de conjuntos heterogéneos de tarefas e pela possibilidade de efectuar operações sobre grupos de tarefas. O *OReK* é um executivo de tempo-real orientado por objectos, completamente preemptivo e implementado em *C++*. Este existia para outras plataformas, nomeadamente para o processador *ARPA* (*Advanced Real-time Processor Architecture*) e para a família *Intel/x86*. De referir que o processador *ARPA* é baseado na arquitectura *MIPS32*, sendo disponibilizado sob a forma de um núcleo sintetizável.

Existia ainda interesse em portar este executivo, por um lado para um processador *soft-core* (sintetizável e com implementação em blocos lógicos configuráveis) optimizado para *FPGA*, por outro lado para um processador de elevado desempenho derivado da sua implementação em recursos dedicados (não programáveis) da *FPGA*. Desta forma torna-se possível uma comparação do executivo sobre três processadores implementados em *FPGA*: Um *soft-core* neutro, um *soft-core* optimizado para *FPGA* e um processador implementado em recursos dedicados dentro da *FPGA* de elevado desempenho.

A escolha do processador *MicroBlaze* deve-se ao facto deste ser optimizado para *FPGAs* da *Xilinx* e para sistemas embutidos. Relativamente ao processador *PowerPC 405*, a sua selecção deve-se às características de elevado desempenho possibilitadas pela sua arquitectura e ainda por estar implementado em recursos dedicados dentro da *FPGA*.

Devido às diferenças internas de cada sistema integrado, tais implementações vão ter desempenhos necessariamente diferentes. Este é um ponto essencial nesta dissertação e que motiva a avaliação e comparação das diversas implementações.

1.4 Objectivos

O principal objectivo deste trabalho é integrar e avaliar um executivo de tempo-real embutido em *FPGA*. O objectivo enunciado pode ser decomposto nos seguintes passos:

- Compreensão dos conceitos subjacentes aos sistemas de tempo-real, sistemas embutidos, sistemas reconfiguráveis, processadores embutidos e sistemas integrados;
- Construção, implementação e verificação dos sistemas integrados *MB-SoC* e *PPC-SoC*,

recorrendo a dispositivos físicos reconfiguráveis e a ferramentas de desenvolvimento baseadas no *software EDK* [Xil08h], disponibilizado pela *Xilinx* para a construção de sistemas embutidos em *FPGA*;

- Desenvolvimento e teste de *hardware* dedicado à medição precisa de intervalos de tempo, com vista a avaliação das arquitecturas e implementações realizadas no âmbito deste trabalho;
- Efectuar o porte e teste do executivo de tempo-real *OReK*, implementado nos sistemas integrados acima referidos;
- Avaliação e estudo comparativo das diversas implementações do executivo *OReK* nas diferentes plataformas.

1.5 Organização da Dissertação

No sentido de estruturar esta dissertação num documento coerente, foram redigidos além deste capítulo introdutório, os seguintes capítulos:

- **Capítulo 2 - Conceitos Fundamentais -** Neste capítulo são apresentados os conceitos fundamentais da dissertação. Começa por introduzir noções sobre sistemas de tempo-real, abordando o conceito de tarefa de tempo-real, escalonamento, comutação de tarefas e acesso a recursos partilhados. Seguidamente são apresentados conceitos essenciais de executivos de tempo-real, sendo discutidos os estados típicos de uma tarefa, a arquitectura genérica, as estruturas e funções de gestão típicas e a caracterização temporal. Por fim, são resumidos alguns trabalhos relacionados mais relevantes.
- **Capítulo 3 - O Executivo de Tempo-real OReK -** Este capítulo descreve as funcionalidades, a arquitectura interna, a implementação e a utilização do executivo *OReK*, portado no âmbito desta dissertação para diversos processadores embutidos em *FPGA*. Por fim, é descrita a sua utilização em aplicações, sendo exibido um exemplo concreto.
- **Capítulo 4 - Avaliação do Desempenho -** Este capítulo apresenta a avaliação do desempenho das arquitecturas concebidas no contexto desta dissertação, com base num estudo comparativo do tempo de execução das diversas funções internas e dos serviços prestados à aplicação pelo executivo de tempo-real *OReK*.
- **Capítulo 5 - Conclusão -** Este capítulo completa a dissertação, apresenta um resumo do trabalho realizado, efectua uma análise final dos resultados obtidos e termina com a exposição de possíveis linhas de trabalho e investigação futura.

De modo a fornecer informação complementar acerca do trabalho realizado no âmbito desta dissertação, foram redigidos os seguintes apêndices:

- **Apêndice A - O Sistema Integrado ARPA-SoC -** Este apêndice descreve o sistema integrado *ARPA-SoC*, mais concretamente, a sua estrutura interna e o fluxo de projecto para aplicações baseadas neste sistema integrado.

- **Apêndice B - Os Sistemas Integrados MB-SoC e PPC-SoC -** Este apêndice apresenta de forma sucinta o *software Xilinx EDK* e os sistemas integrados desenvolvidos *MB-SoC* e *PPC-SoC*, mais concretamente, são descritas as ferramentas de suporte, as arquitecturas dos processadores *MicroBlaze* e *PowerPC 405* e o fluxo de projecto para aplicações baseadas nos sistemas integrados desenvolvidos.
- **Apêndice C - Temporizadores para Avaliação do Desempenho -** Este apêndice descreve o módulo de temporizadores desenvolvidos no âmbito desta dissertação para avaliação do desempenho dos sistemas integrados *MB-SoC* e *PPC-SoC*. São descritas as suas características gerais e os fluxos de *hardware* e de *software*.

Capítulo 2

Conceitos Fundamentais

Sumário

Este capítulo introduz alguns conceitos fundamentais sobre sistemas de tempo-real, começando por apresentar algumas definições, a sua relevância prática e abordando o conceito de tarefa de tempo-real, escalonamento, comutação de tarefas e acesso a recursos partilhados. Seguidamente são apresentados conceitos essenciais de executivos de tempo-real, sendo discutidos os estados típicos de uma tarefa, a arquitectura genérica, as estruturas e funções de gestão típicas e a caracterização temporal. Por fim, são expostos trabalhos relacionados relevantes.

2.1 Sistemas de Tempo-real

Um sistema de tempo-real é em geral um sistema reactivo, isto é, um sistema que recebe continuamente informação do ambiente onde está inserido e actua sobre ele a uma cadência adequada à obtenção do comportamento desejado. Este pode ainda ser visto como um sistema computacional capaz de responder a eventos gerados pelo ambiente onde está inserido, dentro de restrições temporais precisas. Dito de outra forma, o correcto funcionamento destes sistemas não depende unicamente dos resultados computacionais produzidos, mas também do instante em que são produzidos [But05].

É ainda importante salientar que um sistema de tempo-real não é um sistema necessariamente rápido, isto é, as restrições temporais são impostas pelo ambiente em que o sistema opera. Uma execução rápida pode ser vantajosa, porém não é suficiente para garantir um comportamento temporal correcto. Por outro lado, o seu comportamento pode ser implementado através de processos ou tarefas concorrentes, sendo cada uma responsável por um sub-conjunto das funcionalidades desempenhadas pelo sistema.

Ao contrário dos sistemas computacionais gerais (como é o caso dos computadores pessoais), cujo grande objectivo é a redução do tempo médio gasto na execução de um conjunto de tarefas e consequentemente um aumento do desempenho médio, os sistemas de tempo-real devem ter o cuidado de garantir o desempenho, a previsibilidade e a correcção temporal, de forma global e ainda individualmente para cada tarefa.

Um factor crucial inerente aos sistemas de tempo-real é a sua previsibilidade. A experiência tem demonstrado que a simulação e o teste do sistema, apesar de necessários, permitem apenas uma avaliação parcial do seu comportamento, isto é, o teste de um sistema não faz uma verificação integral do seu comportamento. No entanto, um sistema de tempo-real para aplicações críticas deverá deve ser projectado assumindo as condições mais desfavoráveis, ser tolerante a falhas, possuir elevada fiabilidade e ainda ser previsível em situações de sobrecarga.

De referir ainda que em sistemas de tempo-real baseados em sistemas operativos é importante que seja o respectivo núcleo a assegurar a resposta em tempo-real aos eventos, o cumprimento das restrições temporais e a tolerância a falhas.

2.1.1 Modelo de Programação Baseado em Tarefas

Na literatura de sistemas operativos os termos tarefa e processo são frequentemente utilizados com o mesmo sentido. Tarefas ou processos são abstracções que incluem um espaço de endereçamento próprio (possivelmente partilhado), um contexto de execução formado pelo conjunto de registos do processador, além de vários outros atributos cujos detalhes variam de sistema para sistema.

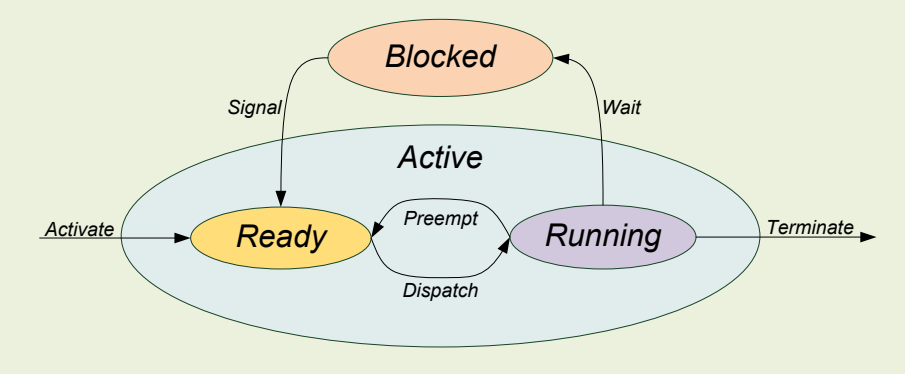


Figura 2.1: Diagrama simplificado de transição de estados de uma tarefa.

Tal como anteriormente referido, os sistemas de tempo-real são geralmente constituídos por um conjunto de tarefas que executam concorrentemente sobre um executivo multi-tarefa. Uma tarefa é ainda uma sequência de activações, cada uma composta por um conjunto de instruções que, na ausência de outras actividades, é executada de forma ininterrupta pelo processador até ser concluída.

De forma simplificada, uma tarefa pode encontrar-se num dos seguintes estados (figura 2.1):

- Pronta a executar (*ready*) - se estiver à espera que lhe seja atribuído tempo da *CPU* (*Central Processing Unit*);
- A executar (*running*) - se estiver a ser executada pela *CPU*;
- Bloqueada (*blocked*) - se estiver a aguardar a libertação de um recurso ou à espera de um evento.

Uma tarefa designa-se por activa se estiver a executar ou pronta a executar. As tarefas activas são armazenadas numa fila de espera chamada *ready queue*. À estratégia utilizada para escolher uma tarefa entre as que se encontram activas e atribuir-lhe tempo de processamento, chama-se algoritmo ou política de escalonamento. A figura 2.1 ilustra ainda os eventos responsáveis pela transição entre os diversos estados que uma tarefa pode estar.

2.1.1.1 Parâmetros

Uma tarefa τ_i é caracterizada por diversos parâmetros (figura 2.2):

- a_i - instante de activação;
- s_i - instante de execução;
- f_i - instante de terminação;
- d_i - *deadline* absoluta;
- D_i - *deadline* relativa;
- C_i - tempo máximo de execução (*Worst Case Execution Time - WCET*);
- $c_i(t_k)$ - tempo máximo de execução residual;
- $x_i(t_k)$ - tempo mínimo de relaxamento (folga);

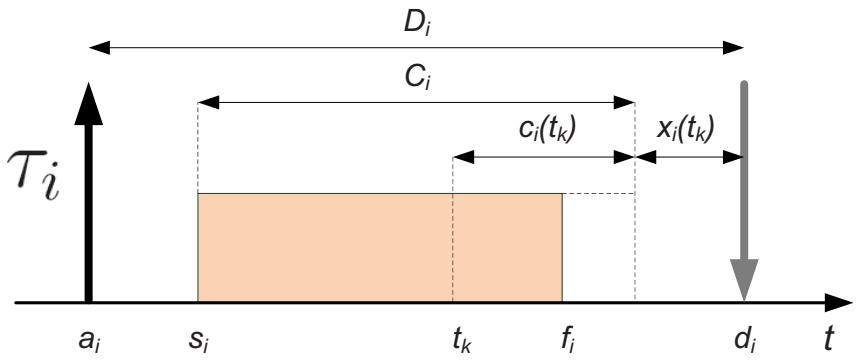


Figura 2.2: Parâmetros temporais de uma tarefa.

De referir que $c_i(t_k)$ é um majorante do tempo de execução restante e $x_i(t_k)$ é um minorante do relaxamento (folga da tarefa) no instante t_k.

Para um dado instante t_k, o tempo mínimo de relaxamento é definido da seguinte forma:

$$x_i(t_k) = d_i - [t_k + c_i(t_k)]$$

Se $x_i(t_k) < 0$, a tarefa pode não conseguir terminar antes de d_i, dito de outra forma, existem fortes possibilidades de ocorrer uma violação temporal (*deadline miss*).

O atraso L_i na terminação de uma tarefa pode ser definido da seguinte maneira:

$$L_i = f_i - d_i$$

- Se $Li > 0$, significa que a tarefa passou o instante de terminação, tendo ocorrido uma perda de *deadline*;
- Se $Li <= 0$, significa que a tarefa concluiu dentro do tempo de terminação.

2.1.1.2 Criticalidade

Quanto à criticalidade, as tarefas podem ser caracterizadas da seguinte forma:

- Tempo-real críticas (*hard real-time*) - se o não cumprimento das restrições temporais acarretar efeitos desastrosos no sistema controlado ou no ambiente;
- Tempo-real não críticas (*soft real-time*) - se o não cumprimento das restrições temporais originar uma degradação do desempenho do sistema;
- Ordinárias (*non real-time*) - se não possuírem restrições temporais.

2.1.1.3 Periodicidade e Modo de Activação

Relativamente à periodicidade e ao modo de activação, as tarefas podem ser caracterizadas da seguinte forma:

- Periódicas - as tarefas são activadas em intervalos de tempo periódicos;
- Esporádicas - a activação das tarefas é efectuada assincronamente quando ocorrer um evento, ou através da invocação explícita de uma primitiva de activação. É ainda necessário que seja respeitado um intervalo mínimo entre activações consecutivas (*Minimum Interarrival Time - MIT*);
- Aperiódicas - podem ser activadas em qualquer instante pelo que só podem ser caracterizadas de forma probabilística.

Refira-se ainda que uma tarefa periódica, além dos paramentos anteriormente expostos, possui um período T_i associado. É ainda importante salientar que a literatura por vezes refere tarefas esporádicas como um sub-conjunto das aperiódicas, diferenciando-as apenas pelo *MIT*.

2.1.1.4 Restrições

As restrições das tarefas podem ser de vários tipos:

- Temporais - definem os diversos parâmetros e restrições temporais das tarefas, tais como instante inicial de activação (fase), período, *deadline*, etc.;
- Precedência - estabelecem uma determinada ordem de execução entre tarefas regulares;
- Utilização de recursos - necessidade de utilização de recursos partilhados com acesso mutuamente exclusivo.

O acesso a recursos partilhados será discutido na subsecção 2.1.3 deste capítulo.

2.1.1.5 Preempção

Uma tarefa admite preempção se puder ser temporariamente interrompida para a execução de outra mais prioritária. Um conjunto de tarefas admite preempção total quando todas as tarefas admitem preempção em qualquer ponto da sua execução. É ainda importante salientar que o acesso a recursos partilhados pode impor restrições no grau de preempção que uma tarefa admite.

2.1.2 Escalonamento de Tarefas

Dito de forma simplificada, o escalonamento é uma atribuição particular de tarefas a um ou mais processadores. O problema do escalonamento baseia-se em determinar uma atribuição de um conjunto recursos, processadores e tarefas, que produza um escalonamento praticável.

Um escalonamento diz-se praticável se satisfaz todas as restrições associadas ao conjunto de tarefas. Um conjunto de tarefas diz-se escalonável se existir pelo menos um escalonamento praticável.

2.1.2.1 Taxonomia

À estratégia utilizada para eleger uma tarefa entre as que se encontram activas e atribuir-lhe tempo de processamento denomina-se por critério, política ou algoritmo de escalonamento.

Os algoritmos de escalonamento podem ser classificados em:

- Preemptivo *versus* não-preemptivo;
- Estático *versus* dinâmico;
- *Off-line versus on-line*;
- Óptimo *versus* sub-óptimo.

Um algoritmo de escalonamento é preemptivo se a tarefa a executar admitir preempção. No caso de a tarefa a executar não admitir preempção, o algoritmo é não preemptivo.

Diz-se que um algoritmo de escalonamento é estático se as decisões de escalonamento forem tomadas com base em parâmetros fixos, atribuídos estaticamente às tarefas antes das sua activação. Por outro lado, se o escalonamento for baseado em parâmetros cujo valor pode variar com o tempo, o algoritmo diz-se dinâmico.

O escalonamento diz-se *On-line* se for feito durante a execução do sistema, para o conjunto das tarefas activas. Ao invés, se a ordem de execução for determinada antes do sistema entrar em funcionamento, isto é, se o escalonamento estiver armazenado numa tabela, o escalonamento diz-se *Off-line*.

Um algoritmo diz-se óptimo se no caso de existir um escalonamento possível, ele conseguir determiná-lo. No caso oposto, o algoritmo designa-se por sub-óptimo.

2.1.2.2 Políticas de Escalonamento para Sistemas de Tempo-real

Os sistemas de tempo-real possuem restrições temporais que devem ser tidas em consideração pelo algoritmo de escalonamento. Mais concretamente, possuem uma *deadline* (tempo limite de execução) que deve ser respeitada sob a pena dos valores produzidos perderem utilidade ou deixarem de ser válidos. Os algoritmos para escalonamento de tarefas de tempo-real baseiam-se especialmente nos parâmetros período, *deadline* relativa e *deadline* absoluta. Nos primeiros dois casos o escalonamento é baseado num parâmetro estático, enquanto que no último caso baseia-se num parâmetro dinâmico.

De referir ainda que nos sistemas operativos convencionais são usadas diversas políticas de escalonamento clássicas como *First Come First Served*, *Round Robin*, *Shortest Job First* e *Priority Scheduling* (com um conjunto muito limitativo de níveis de prioridade).

Estas políticas têm vantagens e desvantagens e visam distribuir de uma forma "justa"o tempo da *CPU* por todas as tarefas do sistema. A escolha de uma política em particular depende do fim em concreto. No entanto, tais políticas não devem ser utilizadas em sistemas de tempo real porque não garantem o cumprimento das restrições temporais impostas às tarefas e consequentemente, não asseguram uma resposta em tempo-real e previsível do sistema.

Escalonamento Cíclico

O escalonamento cíclico (*timeline schedule*) foi o primeiro a ser utilizado no escalonamento de tarefas de tempo-real. O escalonamento é feito *off-line* e consiste no seguinte processo:

O eixo do tempo é dividido em intervalos com duração igual (*time slots*), cada tarefa é alocada estaticamente nas *slots* necessárias de forma a cumprir a periodicidade desejada, e a execução em cada *slot* é activada por um temporizador.

Esta política de escalonamento tem como vantagens:

- Implementação simples, não necessitando de um sistema operativo de tempo-real;
- Sobrecarga computacional reduzida durante a execução, uma vez que todo o escalonamento é feito *off-line*;
- Bom controlo sobre variações do instante de execução (*jitter*) das tarefas.

No entanto, esta política possui as seguintes desvantagens:

- A integração de tarefas aperiódicas é bastante complicada;
- A adição de novas tarefas obriga à reconstrução de todo o escalonamento;
- Não é robusta em situações de sobrecarga.

As desvantagens anteriores são bastante restritivas, razão pela qual foram propostos algoritmos ou politicas de escalonamento baseados em prioridades.

Escalonamento Baseado em Prioridades

O escalonamento baseado em prioridades atribui a cada tarefa uma prioridade calculada a partir das suas restrições temporais. A execução das tarefas é efectuada por um executivo baseado em prioridades.

Este método permite efectuar uma análise da escalonabilidade usando técnicas analíticas. As prioridades são geralmente atribuídas com base nos parâmetros período e *deadline*. Algumas políticas de escalonamento têm ainda em conta o tempo máximo de execução, quer para efeitos de escalonamento, quer para análise de escalonabilidade.

Actualmente, em sistemas de tempo-real são tipicamente empregues as seguintes políticas de escalonamento: *Rate Monotonic* (*RM*), *Deadline Monotonic* (*DM*), *Earliest Deadline First* (*EDF*) e *Least Slack First* (*LSF*).

O algoritmo Rate Monotonic [LL73] atribui a cada tarefa uma prioridade fixa inversamente proporcional ao seu período de activação. No algoritmo *RM* considera-se que $D_i = T_i$, isto é, uma tarefa deve terminar antes da sua próxima activação.

O algoritmo *RM* possui as seguintes características:

- Prioridade fixa inversamente proporcional ao período;
- Activação simultânea de todas as tarefas;
- Minimiza o atraso máximo da tarefa com menor período;
- Preemptivo.

O algoritmo *RM* é ainda considerado óptimo entre os algoritmos de prioridade fixa.

O algoritmo Deadline Monotonic [LW82] é uma generalização do *RM*. Este selecciona para execução a tarefa com *deadline* relativa mais pequena. No algoritmo *DM* considera-se que $D_i <= T_i$.

O algoritmo *DM* possui as seguintes características:

- Prioridade fixa inversamente proporcional à *deadline* relativa;
- Activação simultânea de todas as tarefas;
- Minimiza o atraso máximo da tarefa com menor *deadline* relativa;
- Preemptivo.

O algoritmo Earliest Deadline First [LL73] selecciona para execução a tarefa com *deadline* absoluta mais próxima. Dito de outra forma, a prioridade de uma tarefa é inversamente proporcional ao tempo para a sua *deadline* absoluta.

O algoritmo *EDF* possui as seguintes características:

- Prioridade dinâmica;
- Minimiza o número de mudanças de contexto relativamente aos algoritmos *RM* e *DM*;
- Minimiza o atraso máximo de todas as tarefas;
- Preemptivo.

De referir que em sistemas uni-processador o algoritmo *EDF* é considerado óptimo entre todos os algoritmos de escalonamento. A prioridade dinâmica permite, em cada instante, a atribuição do processador à tarefa com *deadline* absoluta mais próxima. Ao contrário do algoritmo *RM*, permite alcançar uma taxa de utilização do processador de 100%, garantindo a escalonabilidade.

O algoritmo Least Slack First tem em consideração o tempo máximo de execução e a folga de cada tarefa. A cada instante, o algoritmo selecciona para execução a tarefa com o menor tempo de relaxamento (tempo livre). A prioridade das tarefas prontas aumenta com o tempo (prioridade dinâmica). O algoritmo *LSF* é ainda considerado óptimo entre todos os algoritmos de escalonamento.

Está fora do âmbito desta dissertação uma discussão mais exaustiva sobre o escalonamento de tarefas em sistemas de tempo-real. Em [Sha04] encontra-se um resumo de trabalho publicado com enquadramento nesta área.

2.1.3 Acesso a Recursos Partilhados

É frequente as aplicações de tempo-real executarem tarefas concorrentes com acesso a recursos partilhados. Para assegurar a integridade destes recursos e consequentemente, o correcto funcionamento das aplicações, o acesso a tais recursos deve ser efectuado em regime de exclusão mútua. Por outro lado, quando tarefas com diferentes prioridades concorrem pelo acesso a recursos partilhados, podem ocorrer inversões de prioridade (figura 2.3). Ou seja, tarefas de maior prioridade podem ficar à espera (bloqueadas) da libertação de recursos anteriormente adquiridos por tarefas menos prioritárias.

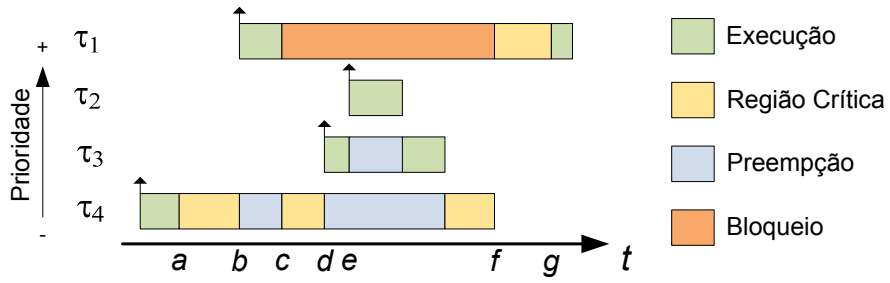


Figura 2.3: O problema da inversão de prioridades no acesso a recursos partilhados.

Considere-se o cenário da figura 2.3 onde é apresentado um conjunto de tarefas τ_1, τ_2, τ_3, τ_4, com prioridades fixas de tal forma que τ_1 é a mais prioritária e τ_4 a menos prioritária.

A tarefa τ_4 começa a executar e entra na região crítica (instante a), adquirindo o recurso partilhado em regime de exclusão mútua. No instante b, τ_1 inicia a sua execução, ficando bloqueada a partir do momento em que requisita o recurso partilhado com a tarefa τ_4 (instante c). No instante d, τ_3 inicia a sua execução, forçando a preempção de τ_4. No instante e, τ_2 inicia a sua execução, forçando igualmente a preempção de τ_3. Desta forma e até ao instante f, τ_1 é bloqueada pela execução da tarefa τ_4 e ainda das tarefas τ_3 e τ_2, com as quais τ_1 não partilha qualquer recurso.

O bloqueio causado por τ_4 designa-se por bloqueio directo enquanto que os causados por τ_3 e τ_2 designam-se por bloqueio indirecto. Este cenário caracteriza o problema da inversão de prioridades e da limitação da sua duração. Note-se que, à semelhança de τ_2 e τ_3, quaisquer tarefas com prioridades entre as de τ_1 e τ_4 poderão efectuar preempção sobre τ_4 e alongar indefinidamente o bloqueio indirecto causado a τ_1.

É importante referir que a inversão de prioridades é um fenómeno que não pode ser completamente evitado na presença de recursos partilhados com acesso exclusivo. Contudo, em sistemas de tempo-real é fundamental determinar a sua duração e limitá-la.

Conforme clarificado no exemplo anterior, as inversões de prioridade surgem associadas à preempção de tarefas. Isto é, num sistema sem preempção, considerando que no final de cada instância de uma tarefa, esta deverá libertar todos os recursos utilizados, pelo que apenas uma tarefa de cada vez pode aceder a cada recurso partilhado. Desta forma o problema da sincronização no respectivo acesso fica implicitamente resolvido.

De referir que as técnicas utilizadas para limitação da inversão de prioridades são, na verdade, técnicas para controlo de preempção. Estas variam em termos de impacto sobre a pontualidade do sistema e da dificuldade de concretização.

Tais técnicas tipicamente podem ser divididas em dois grupos. No primeiro constam as técnicas chamadas básicas, estas são de fácil implementação mas não são selectivas relativamente às tarefas que sofrem bloqueio, isto é, bloqueiam igualmente todas as tarefas de maior prioridade, quer utilizem ou não o recurso partilhado em questão. No segundo grupo encontram-se as técnicas baseadas em semáforos que, embora mais complexas, permitem reduzir o conjunto das tarefas afectadas pelos bloqueios associados às inversões de prioridade.

2.1.3.1 Técnicas Básicas de Sincronização

Como anteriormente referido, as técnicas ditas básicas bloqueiam indistintamente as tarefas, sendo por isso pouco eficientes e causadoras de considerável perturbação na pontualidade do sistema. Como métodos de sincronização básicos encontram-se as técnicas de inibição de interrupções e de inibição de preempção.

Inibição de Interrupções

A inibição de interrupções é uma das formas de inibição de preempção causada pelo executivo. Na verdade, a activação de tarefas de maior prioridade é, regra geral, efectuada ou pelo mecanismo de gestão temporal do executivo (tarefas periódicas) inserido na rotina de atendimento da interrupção periódica do sistema (*tick*), ou por interrupções assíncronas (tare-

fas aperiódicas). Desta forma, enquanto as interrupções estiverem inibidas não poderá haver activação de outras tarefas, pelo que fica garantido o acesso exclusivo a quaisquer recursos.

Para utilizar esta técnica é suficiente desactivar as interrupções quando uma tarefa entra na sua região crítica, tomando posse de um recurso partilhado, seguindo-se a activação das interrupções após a saída. Consequentemente, todas as restantes actividades do sistema ficam bloqueadas, não só as tarefas de maior prioridade, mas também o atendimento a interrupções externas e à interrupção periódica do sistema. Esta última é de grande importância e no caso de o bloqueio ser suficientemente longo, podem perder-se *ticks*, o que origina um funcionamento incorrecto do sistema.

Quanto à duração do bloqueio causado por esta técnica, uma tarefa só pode ser bloqueada uma única vez e por um intervalo de tempo que, no pior caso, é igual à maior região crítica das tarefas de menor prioridade.

Inibição de Preempção

Uma optimização em relação à técnica anterior, consiste em inibir directamente a preempção de tarefas sempre que estas estejam a executar numa região crítica, mas sem inibir o atendimento a interrupções. Esta técnica apesar de fácil concretização, não bloqueia a actividade associada ao atendimento de interrupções, incluindo o atendimento do relógio do sistema, sendo menos restritiva que a anterior. Por outro lado, mantêm-se os problemas relacionados com o bloqueio indistinto de todas as tarefas mais prioritárias, independentemente de usarem ou não quaisquer recursos que estejam a ser acedidos. Relativamente à duração do bloqueio causado, este é semelhante ao da técnica anterior.

2.1.3.2 Técnicas de Sincronização Baseadas em Semáforos

As técnicas de sincronização baseadas em semáforos permitem, ao contrário das anteriores, reduzir o conjunto das tarefas que sofrem bloqueio via inversão de prioridades inerente ao acesso a recursos partilhados. Nestas técnicas, apenas as tarefas que necessitam de aceder a recursos poderão ser bloqueadas. De referir que a utilização de semáforos necessita de um conjunto adicional de recursos (mecanismos e estruturas de dados), sendo necessariamente mais complexa de concretizar e de utilizar. Por outro lado, a redução do número de situações de bloqueio leva a menores tempos de resposta, o que resulta num melhoramento funcional global do sistema.

De uma forma muito simplista, o funcionamento destas técnicas passa pela associação de um semáforo a cada recurso, existindo a necessidade de cada tarefa passar pelo controlo do respectivo semáforo antes de entrar na região crítica. De acordo com o estado do semáforo, a tarefa pode ou não entrar na zona crítica. No caso de não ser facultado o acesso, a tarefa fica bloqueada e é inserida numa fila de espera até que o semáforo autorize e a tarefa tenha prioridade suficiente. De referir que o funcionamento dos semáforos e das respectivas filas de espera é gerido por protocolos específicos.

Relativamente aos sistemas de tempo-real, um protocolo de gestão de semáforos deverá evitar situações de adiamento indefinido (*deadlock*), bloqueios indeterminados e bloqueios em cadeia. Como métodos de sincronização baseados em semáforos típicos em executivos de

tempo-real, encontram-se o Protocolo de Herança de Prioridade (*PIP - Priority Inheritance Protocol*), o Protocolo de Tecto de Prioridade (*PCP - Priority Ceiling Protocol*) e a Política de Pilha de Recursos (*SRP - Stack Resource Policy*), os quais serão descritos resumidamente de seguida.

Protocolo de Herança de Prioridade (*PIP*)

O Protocolo de Herança de Prioridade [Lui90] corresponde a uma solução simples mas eficiente para eliminar o indeterminismo associado ao bloqueio indirecto causado nas tarefas de maior prioridade pelas tarefas de prioridade intermédia. A utilização do Protocolo de Herança de Prioridade aplica-se a tarefas que sejam definidas com prioridade estática, atribuída por sistemas de prioridade fixa (ex. *RM*, *DM*).

Na execução de um sistema com o protocolo *PIP* implementado, sempre que uma tarefa está a bloquear outra de maior prioridade, a sua prioridade é aumentada provisoriamente, herdando o valor da tarefa bloqueada e reduzindo assim a possibilidade de preempção da tarefa bloqueante. Na ausência de acessos encadeados a recursos partilhados, cada tarefa pode somente bloquear outra uma única vez, e só pode ficar bloqueada uma vez em cada semáforo.

Desta forma, uma tarefa mais prioritária pode sofrer dois tipos de bloqueios [But97]:

- Bloqueio directo: ocorre quando a tarefa mais prioritária tenta aceder a um recurso partilhado que se encontra bloqueado por uma tarefa menos prioritária.
- Bloqueio por herança: ocorre quando uma tarefa de prioridade intermédia é impedida de continuar a sua execução por uma tarefa que tenha herdado a prioridade de uma tarefa mais prioritária.

É ainda importante referir que o protocolo *PIP*, além de possuir maior complexidade, apresenta alguns efeitos graves no acesso encadeado a recursos partilhados, nomeadamente a possibilidade de ocorrência de bloqueios em cadeia e de adiamento indefinido.

Protocolo de Tecto de Prioridade (*PCP*)

O Protocolo de Tecto de Prioridade [Lui90] tem como principais objectivos limitar o número de bloqueios por inversão de prioridades, evitar a ocorrência de bloqueios em cadeia e *deadlocks* durante a execução das tarefas. Este protocolo é uma extensão do *PIP* ao qual se adicionou uma regra de controlo no acesso aos semáforos livres baseada no conceito de tecto de prioridade. À semelhança do protocolo *PIP*, o *PCP* é também dirigido a sistemas de prioridades fixas.

Nas implementações com o protocolo *PCP*, todos os recursos (semáforos) acedidos em regime de exclusão mútua possuem um valor de prioridade tecto (*ceiling priority - $C(S_k)$*) que corresponde à prioridade mais elevada do conjunto de tarefas que acedem ao recurso.

Para entrar na região crítica é necessário que o recurso partilhado esteja livre e a prioridade da tarefa seja maior que a prioridade tecto (*ceiling*) de qualquer recurso que nesse momento esteja bloqueado.

Esta particularidade do protocolo *PCP* serve para garantir que quando uma tarefa acede a um recurso partilhado, todos os recursos de que poderá ainda necessitar durante a sua execução estão livres. Desta forma demonstra-se que o protocolo evita *deadlocks*, e permite constatar que para além do bloqueio indirecto do tipo *push-through* inerente à herança de prioridades, com o protocolo *PCP* uma tarefa só pode ser bloqueada uma vez e logo no seu primeiro acesso a um recurso partilhado.

É ainda importante referir que a duração máxima de bloqueio que uma tarefa pode sofrer corresponde à duração da maior região crítica das tarefas de menor prioridade que partilham recursos com a tarefa em questão ou com outras tarefas de maior prioridade.

Política de Pilha de Recursos (*SRP*)

A Política de Pilha de Recursos [Bak91] consiste num protocolo desenvolvido para poder ser aplicado indistintamente em sistemas de prioridades fixas ou dinâmicas. A sua principal característica é o facto de apenas impor bloqueios antes das tarefas iniciarem a execução. Após o início de execução e até à terminação, as tarefas não sofrerão qualquer bloqueio. Refira-se o antagonismo face aos protocolos *PCP* e *PIP*, com os quais uma tarefa só é bloqueada quando tenta entrar numa região crítica.

No protocolo *SRP*, a cada tarefa τ_i é atribuído um nível de preempção π_i. Este é um parâmetro estático, cujo significado pode ser interpretado como a capacidade de uma tarefa efectuar preempção sobre outras. Em sistemas de prioridades fixas, o nível de preempção corresponde à prioridade. Para sistemas de prioridades dinâmicas ditadas pela proximidade à *deadline*, tal como no algoritmo de escalonamento *EDF*, o nível de preempção é definido na ordem inversa das *deadlines* relativas.

À semelhança do protocolo *PCP*, o *SRP* atribui a cada semáforo R_k uma prioridade tecto $C(R_k)$ correspondente ao maior nível de preempção de entre todas as tarefas que usam esse semáforo. O protocolo *SRP* define ainda a prioridade tecto do sistema (*system ceiling* - $\Pi(t)$) que é a maior prioridade tecto de todos os semáforos que estão fechados num dado instante:

$\Pi(t) = max[C(R_k) : R_k$ está fechado no instante $t]$

Neste caso, uma tarefa quando activada apenas poderá iniciar a execução se tiver a prioridade adequada e o seu nível de preempção for, nesse instante, superior ao tecto do sistema. Com um raciocínio análogo ao efectuando no caso do protocolo anterior, facilmente se demonstra que nessas circunstâncias os recursos de que uma tarefa necessita estão todos livres e a tarefa executará sem bloqueios.

De referir que o instante de bloqueio das tarefas é feito na preempção, facto que assegura a ausência de *deadlocks* e de bloqueios em cadeia e, por outro lado, permite reduzir o número de preempções e de comutações de contexto, pelo que efectivamente causa menor sobrecarga computacional no sistema.

Está fora do âmbito desta dissertação uma discussão mais exaustiva sobre os protocolos de sincronização em sistemas de tempo-real. Esta subsecção resumiu alguns conceitos funda-

mentais sobre sistemas de tempo-real. A informação apresentada foi essencialmente reunida de [But97, But05] e [AP07], onde podem ser encontrados exemplos ilustrativos de cada um dos protocolos.

2.2 Executivos de Tempo-real

Actualmente, devido à crescente complexidade e tempo de desenvolvimento, a generalidade das aplicações são construídas a partir de serviços oferecidos por um sistema operativo. No caso das aplicações de tempo-real, o cumprimento de requisitos temporais depende não só do código da aplicação, mas também do comportamento do sistema operativo no sentido de permitir previsibilidade ou pelo menos um desempenho regular. Muitas vezes os requisitos temporais da aplicação são tão rigorosos que o sistema operativo é substituído por um executivo de tempo-real. Os executivos de tempo-real oferecem apenas as funcionalidades essenciais, porém são capazes de apresentar um comportamento temporal excelente, possibilitado pela sua simplicidade interna.

A utilização de um executivo permite, além de um controlo rigoroso de todo o sistema, simplificar o desenvolvimento de aplicações. De uma maneira geral, esta simplificação é tanto maior quanto o grau de complexidade do sistema a desenvolver.

Cabe às aplicações utilizar os serviços disponibilizados pelo executivo. Desta forma, o responsável pelo desenvolvimento da aplicação não precisa preocupar-se com a gestão de recursos básicos e utiliza as abstracções de alto nível criadas pelo executivo (tarefas e mecanismos de comunicação e sincronização entre estas, e em certos casos até gestão de memória, serviço de rede e sistemas de ficheiros). Esta secção discute aspectos fundamentais referentes a executivos de suporte a aplicações de tempo-real.

2.2.1 Estados de uma Tarefa

As tarefas utilizadas em executivos de tempo-real, descritas na secção 2.1.1, possuem os seguintes estados (figura 2.4):

- Dormente (*idle*) - se estiver a aguardar a activação;
- Pronta a executar (*ready*) - se estiver à espera que lhe seja atribuído tempo de *CPU* (*Central Processing Unit*);
- A executar (*running*) - se estiver a ser executada pela *CPU*;
- Suspensa (*suspended*) / Adormecida (*sleeping*) - se estiver à espera de um evento (tipicamente temporal);
- Bloqueada (*blocked*) - se estiver a aguardar um evento ou a libertação de um recurso.

A figura 2.4 ilustra os diversos estados de uma tarefa e ainda os eventos responsáveis pela transição entre estados.

Descrevendo de forma simplificada os estados de uma tarefa, esta começa por ser criada e permanece no estado dormente à espera de um evento de activação. No caso de uma

tarefa periódica, tal evento é despoletado em intervalos de tempo regulares, conferindo a periodicidade desejada para a reactivação de cada instância da tarefa. No caso de uma tarefa aperiódica, o evento de activação é despoletado por intermédio de um sinal externo (tipicamente uma interrupção), ou ainda por primitivas de activação explícita de uma tarefa aperiódica, opcionalmente disponibilizadas pelo executivo.

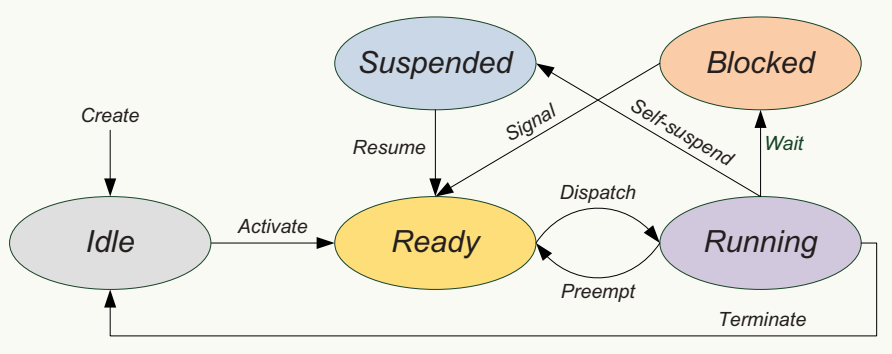


Figura 2.4: Diagrama de transição de estados de uma tarefa.

Após a activação, a tarefa está pronta para ser executada pelo processador. Entretanto, como possivelmente várias tarefas prontas disputam pela atribuição de tempo de processamento, a tarefa pode ficar em fila de espera até ser seleccionada pelo escalonador para entrar em execução. A fila é ordenada por um determinado critério de escalonamento, não sendo (em sistemas de tempo-real) normalmente por ordem de chegada. O evento *dispatch* é responsável por colocar a tarefa em execução.

Uma vez em execução, a tarefa pode encontrar situações de bloqueio ao tentar aceder a recursos partilhados com acesso mutuamente exclusivo, previamente bloqueados por outra tarefa. Nesta situação a tarefa deixa de ser executada pelo processador e passa para o estado bloqueada. Quando a causa de bloqueio desaparece, isto é, o acesso ao recurso partilhado é libertado, a tarefa transita para o estado pronta a executar.

A figura 2.4 diferencia ainda um tipo especial de bloqueio. Nesta situação em particular a tarefa solicita a sua auto-suspensão por um determinado intervalo de tempo. Neste caso, a tarefa encontra-se suspensa. Esta voltará a reatar a execução mais tarde, ainda durante a mesma instância. Por fim, quando uma tarefa termina a sua execução, esta transita para o estado dormente, aguardando a activação da próxima instância.

Note-se que a figura 2.4 descreve o funcionamento típico de um executivo de tempo-real genérico. Uma descrição exacta da semântica de cada estado depende de detalhes que, na prática, variam de executivo para executivo. Por exemplo, uma tarefa nunca ficará bloqueada, no caso de um escalonamento que garanta que estão disponíveis todos os recursos que uma tarefa requeira no momento da sua execução.

2.2.2 Arquitectura Genérica

Os executivos de tempo-real, sendo especializados para proporcionar um comportamento temporal preciso, não incluem funcionalidades acessórias comuns em sistemas operativos, ou disponibilizam com restrições ou implementações modificadas. Por outro lado, providenciam a gestão temporal, de tarefas e de recursos partilhados e ainda o escalonamento e despacho de tarefas:

- Gestão temporal - tipicamente as aplicações necessitam de realizar operações que dependem da passagem do tempo (ex. activações periódicas), além disso, permite realizar o policiamento temporal do sistema para assegurar o seu correcto funcionamento;
- Gestão de tarefas - normalmente proporcionada por chamadas de sistema (*system calls*) ou por funções internas, permite criar e destruir tarefas, realizar as transições de estado indicadas na figura 2.4 e consultar o estado actual. Permite também, de acordo com o algoritmo de escalonamento utilizado, seleccionar a tarefa a executar e ainda a sua colocação em execução (despacho da tarefa);
- Gestão de recursos partilhados - possibilita a comunicação e a sincronização entre tarefas, recorrendo a mecanismos que implementam o acesso em regime de exclusão mútua a recursos partilhados.

A figura 4.3 ilustra a arquitectura genérica de um executivo de tempo-real. Esta será abordada em maior detalhe na próxima subsecção.

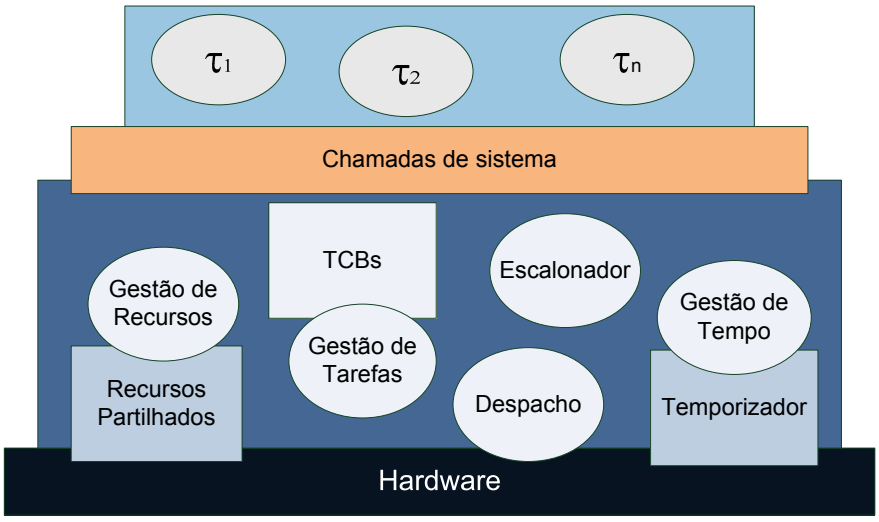


Figura 2.5: Arquitectura genérica de um executivo de tempo-real (adaptado de [AP07]).

2.2.3 Estruturas e Funções de Gestão Típicas

Um executivo de tempo-real tipicamente possui diversas estruturas de dados. Estas facilitam a organização interna e são utilizadas para armazenar informação do executivo e das tarefas.

O executivo fornece ainda um conjunto de funções que, por um lado facilitam o funcionamento interno, por outro lado, proporcionam um conjunto de serviços à aplicação com base na funcionalidade interna e na abstracção que fazem do *hardware*.

2.2.3.1 A Estrutura Task Control Block

A estrutura *TCB* (*Task Control Block*) é fundamental num executivo de tempo-real. Esta serve para caracterizar a tarefa e ainda para gerir a respectiva execução. Segue-se a apresentação de alguns campos usuais:

- Identificador;
- Ponteiro para o código a ser executado;
- Ponteiro para a pilha (*stack*) privada (necessária para a salvaguarda do contexto e armazenamento das variáveis locais);
- Atributos de activação (periódica, esporádica, aperiódica, *single shot*);
- Atributos de criticalidade (crítica, não crítica ou ordinária);
- Outros atributos (*deadline*, prioridade)
- Estado dinâmico de execução e outras variáveis para controlo de activação (temporizadores por *software*, *deadline* absoluta)

De referir que cada tarefa está caracterizada num único *TCB*. Tipicamente, os *TCBs* estão definidos num array estático, o que torna o sistema mais previsível, mas estruturados segundo uma ou várias listas ligadas para facilitar pesquisas sobre o conjunto das tarefas. Qualquer lista ordenada (ex. lista de tarefas prontas - *ready queue*) pode ser facilmente realizada sobre a estrutura de *TCBs*, através de um ponteiro para o próximo *TCB* contido na lista.

2.2.3.2 Gestão Temporal

A gestão do tempo num executivo de tempo-real é fundamental. Esta pode servir para diversos propósitos como activar tarefas periódicas, verificar o cumprimento de restrições temporais, medir intervalos de tempo (inclusivé para auto-suspensão), entre outros.

A gestão é efectuada com recurso a um temporizador do sistema. Tipicamente, este é programado para gerar interrupções periódicas (*ticks*) e a respectiva rotina de serviço (*handler*) faz a gestão do tempo. É ainda importante referir que os atributos temporais do sistema são múltiplos inteiros do *tick*.

Em sistemas cujo funcionamento é baseado em interrupções periódicas, a duração do *tick* estabelece a sua resolução temporal. De facto, quanto menor for o *tick*, melhor será a resolução temporal do sistema. Porém, a gestão do tempo, ou seja, o atendimento do *tick*, acarreta uma sobrecarga computacional que será tanto maior quanto melhor for a resolução temporal. Dito de outra forma, quanto maior for o *tick*, menor será o *overhead* do executivo.

De referir ainda que o executivo é responsável por manter uma variável que conta o número de *ticks* desde o momento da respectiva activação. É de grande importância que a variável

possua uma dimensão adequada à longevidade e resolução temporal do sistema.

A título de exemplo, uma variável de $32bit$ e um *tick* de $1ms$ permite o funcionamento contínuo do executivo sem *wrap around* da variável contadora, durante $\approx 49,7$ dias. Dependendo da função a desempenhar pelo sistema, esta longevidade pode não ser aceitável e a solução pode passar pelo aumento de tamanho da variável contadora ou pelo relaxamento da resolução temporal.

O mesmo exemplo, mas agora com uma variável de $64bit$ funcionaria sem *wrap around* durante $> 584 \times 10^6$ anos. De referir que o *wrap around* do contador leva a ambiguidades no funcionamento de um executivo pelo que não é, de todo, um evento desejável.

2.2.3.3 Acesso a recursos partilhados

Os recursos partilhados com acesso mutuamente exclusivo, tal como o processador, têm de ser geridos de forma apropriada. De facto, tais recursos só podem ser acedidos por uma tarefa de cada vez. Para efectuar este controlo, é comum recorrer a *flags* atómicas (*mutexes*), semáforos ou monitores. No caso de se utilizarem semáforos, deverá existir uma estrutura por semáforo que indique o respectivo estado, e ainda a lista de tarefas a aguardar acesso - *SCB* (*Semaphore Control Block*).

2.2.4 Caracterização Temporal

Os aspectos temporais são uma das características mais importantes de um executivo de tempo-real. Estes estão relacionados com a capacidade do executivo fornecer os mecanismos e as propriedades necessárias para o cumprimento de restrições temporais.

Dado que tanto a aplicação como o executivo partilham os mesmos recursos de *hardware*, o comportamento temporal do executivo afecta necessariamente o comportamento temporal da aplicação. A simples operação de solicitar um serviço ao executivo através de uma chamada ao sistema, significa que o processador passa a estar ocupado por funções do executivo, pelo que não poderá executar a aplicação. Além disso, a capacidade da aplicação cumprir a respectiva *deadline* passa a depender da prontidão do executivo em fornecer o serviço solicitado no menor intervalo de tempo possível, de forma a não inviabilizar as *deadlines*.

Finalmente, em relação ao comportamento temporal do sistema, qualquer análise deve considerar o conjunto formado pela aplicação e executivo.

2.2.4.1 Métricas de Avaliação

Uma métrica bastante utilizada é o tempo de comutação entre duas tarefas. Este tempo inclui a salvaguarda de registos (contexto) da tarefa que estava em execução e o restauro de contexto da nova tarefa a executar. De uma maneira geral, esta métrica não inclui o tempo necessário para decidir qual tarefa a executar, uma vez que isso depende do algoritmo de escalonamento utilizado.

Outra métrica característica de um executivo é a latência até ao início do atendimento de uma interrupção do *hardware*. Em muitas situações eventos de elevada prioridade e urgentes no sistema são sinalizados por interrupções de *hardware*. Desta forma, é importante iniciar

rapidamente o atendimento destas interrupções. Refira-se que, no caso mais simples, é suposto que o respectivo *handler* seja capaz de gerar uma resposta apropriada ao evento sinalizado.

A latência inclui o tempo necessário para realizar o processamento de uma interrupção, isto é, salvaguarda de contexto e execução do respectivo *handler*. É ainda necessário incluir o tempo máximo durante o qual as interrupções podem ficar desactivadas. Este último é particularmente importante em executivos que desactivam as interrupções no decorrer das suas funções internas. Neste caso, a latência deve incluir a maior sequência de instruções internas que podem ser executadas, tipicamente uma chamada ao sistema completa.

Uma terceira métrica importante para qualquer executivo é o tempo de execução de cada uma das chamadas ao sistema suportadas (ou funções disponibilizadas). Infelizmente, estes tempos de execução dependem muitas vezes do estado interno do executivo, ou seja, do instante em que a chamada ao sistema é efectuada, da carga do sistema nesse instante e do estado das tarefas.

Para calcular o tempo de resposta do sistema no pior caso é necessário ser pessimista e considerar que o executivo encontra-se no estado que resulta em maior tempo de execução possível para a respectiva chamada. É ainda importante salientar que quanto mais complexo for o executivo, mas difícil será calcular (ou obter) estes tempos. Além disso, eles dependem da arquitectura onde o sistema executa.

De referir ainda que existem diversas outras métricas para a caracterização de um executivo. Estas reflectem a prática da construção de aplicações tempo-real nas últimas décadas, e muitas vezes estão mais ligadas ao desempenho do que ao cumprimento de restrições temporais.

2.2.4.2 Relação entre Métricas e Tempo de Resposta

As métricas como o tempo de comutação de contexto e latência até ao início do atendimento de uma interrupção são úteis na medida em que, quanto menor for o seu valor para um dado executivo, tanto melhor. Por outro lado, estas não são as únicas responsáveis pelos tempos de resposta do sistema. É importante salientar que as diversas tarefas e interrupções do sistema causam interferências que devem ser contabilizadas. Por sua vez, os conflitos decorrentes de recursos partilhados, tanto ao nível de aplicação como ao nível do executivo, causam situações de bloqueio e de inversão de prioridades. Estas contribuem de forma negativa para os tempos de resposta do sistema. Todos estes factores devem ser equacionados e somente com um teste de escalonabilidade adequado (que tenha em conta o critério de escalonamento e o protocolo de gestão de semáforos utilizados) será possível determinar se as restrições temporais serão ou não cumpridas.

Está fora do âmbito desta dissertação uma discussão mais exaustiva sobre a caracterização de executivos de tempo-real. Esta subsecção resumiu alguns conceitos fundamentais sobre executivos de tempo-real. A informação apresentada foi essencialmente reunida de [FOF00] e [AP07].

2.3 Trabalho Relacionado

Esta secção aborda diversos executivos de tempo-real para sistemas embutidos e resume as suas principais características. É ainda importante referir que esta discussão é apenas referente a executivos e a pequenos sistemas operativos de tempo-real que possuem características comparáveis. Sistemas operativos de tempo-real mais complexos como o *Windows Embedded CE* [Mic07] não serão incluídos nesta discussão.

A tabela 2.1 resume de forma metódica o trabalho realizado na área de executivos de tempo-real. A caracterização é feita sobre determinados critérios indicados no topo de cada coluna. A tabela encontra-se ainda ordenada de forma alfabética, relativamente ao nome do executivo.

Tabela 2.1: Quadro resumo de diversos executivos de tempo-real.

Nome do executivo	Processadores suportados	Tamanho do executivo (min/max) (KB)	Memória mínima para funcionamento do executivo (Byte)	Memória mínima requerida por tarefa (Byte)	Níveis de prioridade para as tarefas	Suporte de tarefas críticas	Latência típica na comutação entre tarefas (μs)	Latência máxima no atendimento à interrupção (μs)	Resolução temporal do sistema (ms)	Mecanismos de limite de inversão de prioridades	Políticas de escalonamento	Suporte de unidade de gestão de memória (*MMU*)	Comentários
AVIX [AVI06]	*Microchip ds-PIC30F, ds-PIC33F, PIC24F e PIC24H*	10,0/ 10,0	500	150	> 128	S	5,0	*0*	> 0,1	*PIP*	*Prioritized Round-Robin*	N	Executa permanentemente com as interrupções activas. Disponibiliza semáforos, *mutexes* e mensagens. Latência obtida com processador de 40 *MIPS*.
C EXECUTIVE [INC08]	*x86, ARM, MIPS*, outros.	5/ 22	1000	300	32K		3,0	2,0		S	*Prioritized FIFO, time slice*	N	Possui 95% de código em linguagem portável *ANSI C*. Latência obtida com processador a 100*MHz*.

contínua na próxima página...

...continuação da página anterior

Nome do executivo	Processadores suportados	Tamanho do executivo (min/max) (KB)	Memória mínima para funcionamento do executivo (Byte)	Memória mínima requerida por tarefa (Byte)	Níveis de prioridade para as tarefas	Suporte de tarefas críticas	Latência típica na comutação entre tarefas (μs)	Latência máxima no atendimento à interrupção (μs)	Resolução temporal do sistema (ms)	Mecanismos de limite de inversão de prioridades	Políticas de escalonamento	Suporte de unidade de gestão de memória (*MMU*)	Comentários
embOS [Mic08]	*x86, PowerPC, ARM, MIPS, NEC K0, NIOS*, outros.	1,2/ 4,5	28,5	70	255				$>$ 0,1	N	*Prioritized Round-Robin*	N	Desenhado para ser a base para desenvolvimento de aplicações em sistemas embutidos de tempo-real. Optimizado para minimizar o consumo de memória. Interrupções de alta prioridade nunca são desactivadas.
FreeRTOS [Fre08]	*x86, ARM, MIPS, MicroBlaze*, outros.	2,0/ 5,0	200			S				S	*Prioritized Round-Robin*	N	Executivo de tempo-real *open source*, possui certificação para aplicações de segurança críticas. Licença gratuita.
Nimble [Sol08]	*ARM*	1,4/ 3,9	300	150	32		1,6	1,1		N	*Prioritized FIFO, Prioritized Round-Robin , Time slice Round-Robin*	S	Desenhado para possuir um tamanho reduzido e bom desempenho. Foi dada especial atenção ao escalonador, que possui uma latência de 25 ciclos de relógio. Latência obtida com processador *ARM 7* a 40*MHz*.
QNX Neutrino [Sys08]	*x86, PowerPC, MIPS, StrongARM, SH4*	64/ 64			64					*PIP*	*Prioritized FIFO, Prioritized Round-Robin*	S	Suporta o *standard POSIX 1(a,b,c,d)*. Protecção de memória. Escalável.

continua na próxima página...

...continuação da página anterior

Nome do executivo	Processadores suportados	Tamanho do executivo (min/max) (KB)	Memória mínima para funcionamento do executivo (Byte)	Memória mínima requerida por tarefa (Byte)	Níveis de prioridade para as tarefas	Suporte de tarefas críticas	Latência típica na comutação entre tarefas (μs)	Latência máxima no atendimento à interrupção (μs)	Resolução temporal do sistema (ms)	Mecanismos de limite de inversão de prioridades	Politicas de escalonamento	Suporte de unidade de gestão de memória (*MMU*)	Comentários
ThreadX [Log08]	*x86, PowerPC, ARM, MIPS, NIOS-II, MicroBlaze*, outros.	2/ 30	1k	200	32		2	2		*PIP*	*Prioritized FIFO*, *Prioritized Round-Robin*, *Preemption-Threshold*, *Time-slicing*	N	Arranque rápido: Requer apenas 300 ciclos para inicialização. Disponibiliza semáforos, *mutexes*, flags de eventos. Latência obtida com processador a 40*MHz*.
XMK [SR04]	*Hitachi H8s, H8300. Atmel AVR*	0,3/	18	3	256	N				N	*Prioritized FIFO*	N	Executivo de tempo-real preemptivo, projectado para minimizar o tamanho ocupado e a memória requerida. Suporta microcontroladores com arquitecturas de 8 a 32 bit.
ReTMiK [AGP03]	*x86, 80C188*	3/				S					*Rate-Monotonic*, *Deadline-Monotonic*, *Earliest Deadline First*	N	Desenvolvido para simplificar a construção de aplicações multi-tarefa com recursos mínimos de *hardware*.
velOSity [Sof08]	*x86, PowerPC, ARM, MIPS*, outros.		3K							S	*Prioritized FIFO*, *Prioritized Round-Robin*	N	Disponibiliza um conjunto diversificado de serviços, *drivers* e *BSPs*. Optimizado para tamanho e velocidade de execução.

contínua na próxima página...

...continuação da página anterior

Nome do executivo	Processadores suportados	Tamanho do executivo (min/max) (KB)	Memória mínima para funcionamento do executivo (Byte)	Memória mínima requerida por tarefa (Byte)	Níveis de prioridade para as tarefas	Suporte de tarefas críticas	Latência típica na comutação entre tarefas (μs)	Latência máxima no atendimento à interrupção (μs)	Resolução temporal do sistema (ms)	Mecanismos de limite de inversão de prioridades	Politicas de escalonamento	Suporte de unidade de gestão de memória (*MMU*)	Comentários
VxWorks [Win07]	*x86*, *PowerPC*, *ARM*, *MIPS*, outros.				256	S				*PIP*	*Preemptive*, *Round-Robin*	S	Executivo de tempo-real amplamente utilizado na indústria. Certificado de utilização para aplicações críticas.
$\mu C/OS$-*II* [Mic92]	*x86*, *ARM*, *PowerPC*, *MIPS*, *NIOS*, *SHARC*, *Micro-Blaze*, outros.	5/ 20	300	64	64	S	1	5	1	*PCP*	*Preemptive*	N	Disponibiliza semáforos, flags de eventos, troca de mensagens por mailbox. Certificado de utilização para aplicações críticas.

Da análise do quadro de resumo apresentado na tabela 2.1, constata-se que os vários executivos têm considerações em comum, como o caso do desempenho, tamanho e ocupação em memória. De facto, devido às capacidades de memória e processamento dos sistemas embutidos serem tipicamente reduzidas, estes são factores que pesam grandemente na escolha de um executivo. Outra característica comum é a diversidade de processadores suportados, destacando-se os processadores da família *x86*, *PowerPC*, *MIPS* e *ARM*.

Por fim, refira-se que muitos dos executivos têm ainda em comum os algoritmos de escalonamento *Prioritized FIFO* e *Prioritized Round-Robin*. Estes são, por um lado relativamente simples de implementar, o que contribui para uma redução de tamanho e tempo no escalonamento, por outro lado não são os mais indicados para o cumprimento das restrições temporais impostas às tarefas críticas, sendo o *EDF* uma alternativa possível. O executivo *OReK*, além de utilizar a politica de escalonamento *EDF* para as tarefas de tempo-real críticas, possui sincronização baseada em *SRP*, razões que reforçam a sua utilização no âmbito deste trabalho.

Capítulo 3

O Executivo de Tempo-real OReK

Sumário

Este capítulo apresenta o executivo *OReK* (*Object-oriented Real-time Kernel*). Começa por uma breve introdução, seguida da exposição das suas funcionalidades, arquitectura interna e implementação. Por fim, é descrita a sua utilização em aplicações, sendo exibido um exemplo concreto.

A informação apresentada neste capítulo foi baseada em [Arn07]. Com o consentimento e a pedido do autor, a informação aí contida foi extendida de forma a construir um documento coerente que inclua a descrição das novas funcionalidades e plataformas suportadas pelo executivo *OReK*. Por este motivo foram também mantidas as referências ao processador *ARPA-MT* e ao seu coprocessador *Cop2-OSC* de suporte ao sistema operativo de tempo-real. São ainda indicadas as figuras que foram reutilizadas.

3.1 Introdução

O *OReK* é um executivo de tempo-real orientado por objectos completamente preemptivo e implementado maioritariamente em *C++*.

Tal como já foi mencionado no capítulo anterior, um executivo é uma entidade ou um módulo de *software* responsável pela execução concorrente de tarefas ou processos. As funções normalmente realizadas por um executivo são o escalonamento, o lançamento em execução, a comutação, a terminação, a comunicação e a sincronização de tarefas. Para as duas últimas funções e para controlar o acesso a recursos partilhados, os executivos disponibilizam tipicamente primitivas do tipo semáforos, eventos, mensagens, ou outros com funcionalidades análogas.

Em sistemas simples é comum integrar num único módulo executável, isto é, de forma monolítica, o executivo e as tarefas que constituem o sistema. Em sistemas mais complexos que necessitem, por exemplo, de carregamento dinâmico de tarefas ou processos, de gestão hierárquica de memória, de serviços de rede ou de dispositivos de entrada/saída sofisticados é usual utilizar-se um sistema operativo, do qual o executivo é uma das peças fundamentais.

Tal como já foi referido no capítulo 2, um sistema de tempo-real é normalmente um sistema de controlo reactivo que responde continuamente a eventos produzidos pelo ambiente em que

está inserido. A resposta é feita de acordo com uma estratégia predefinida e cumprindo as restrições temporais definidas. A execução do sistema é despoletada por eventos que podem ser síncronos (periódicos) ou assíncronos (aperiódicos) e provenientes de várias fontes, tais como um temporizador ou um sensor. Em qualquer dos casos, a resposta ao evento é feita através da execução de uma tarefa ou processo, onde o evento é processado e desencadeada a respectiva reacção.

Um executivo de tempo-real é um executivo capaz de gerir tarefas de tempo-real, isto é, com restrições temporais precisas, destinando-se portanto a sistemas de tempo-real. Um executivo de tempo-real tem a responsabilidade de executar as tarefas cumprindo as restrições temporais do sistema. A utilização de executivos em sistemas de tempo-real para fazer a gestão de tarefas tem a vantagem de simplificar o desenvolvimento, de os tornar mais robustos e de proporcionar uma abstracção do *hardware*, tornando-os independentes da plataforma, promovendo assim a portabilidade.

O executivo *OReK* descrito neste capítulo teve a sua origem no executivo *ReTMiK* [AGP03]. Relativamente ao *ReTMiK*, foram várias as alterações e melhoramentos introduzidos, nomeadamente:

- A conversão do código fonte escrito em linguagem *C* para *C++* e adopção do paradigma de orientação por objectos;
- A gestão de tarefas com base em listas bi-ligadas de forma a optimizar a sua manipulação;
- O suporte para conjuntos heterogéneos de tarefas incluindo as de tempo-real críticas e não críticas, quer periódicas quer aperiódicas e tarefas ordinárias de baixa prioridade;
- A capacidade de gestão de tarefas em grupos definidos pela aplicação;
- A adição de primitivas de controlo da preempção das tarefas e de sincronização baseada em semáforos com limitação do tempo de bloqueio em que ocorrem fenómenos de inversão de prioridade nas tarefas em execução;
- A inclusão dos mecanismos que permitam tirar partido das capacidades de multi-tarefa simultânea do processador *ARPA-MT* [Arn07].

3.1.1 Plataformas Suportadas

A generalidade do executivo *OReK* é independente do processador alvo, possuindo apenas segmentos bem localizados de código específico da plataforma. Actualmente pode ser utilizado em *PCs* com processador compatível com a família *Intel x86*, no processador *ARPA-MT* e nos sistemas integrados *MB-SoC* e *PPC-SoC*, os quais incluem os processadores *MicroBlaze* e *PowerPC 405*, respectivamente.

No primeiro caso, o executivo *OReK* deve ser executado directamente sobre *MSDOS*, uma vez que necessita de configurar e aceder ao *hardware* do *PC*, mais concretamente ao temporizador e ao controlador de interrupções. No caso da plataforma *ARPA-MT*, o executivo *OReK* pode funcionar total ou parcialmente em *software*, sendo na segunda hipótese suportado pelo coprocessador *Cop2-OSC* (*Cop2 - Operating System Coprocessor*) [Arn07]. Por último, no

caso dos sistemas integrados *MB-SoC* e *PPC-SoC*, o executivo *OReK* opera completamente em *software*. De referir ainda que a implementação do executivo *OReK* em processadores da familia *Intel x86* não possui relevância de ordem prática para este trabalho. Esta foi a implementação inicial, não sendo orientada para sistemas integrados em *FPGA*, pelo que não será incluída nesta discussão.

O executivo *OReK* implementa uma camada de abstracção, fornecendo um conjunto de serviços para gestão de tarefas e semáforos com restrições temporais através de uma interface independente da plataforma utilizada (inclusivamente da implementação do coprocessador *Cop2-OSC*) [Arn07]. Se este coprocessador estiver implementado, o executivo *OReK* usa as funcionalidades por ele disponibilizadas, caso contrário executa todas as suas funções completamente em *software*.

O executivo *OReK* foi desenvolvido em *C++*, fornecendo uma interface de programação orientada por objectos. É disponibilizado na forma de uma biblioteca destinada a ser ligada com o código da aplicação, obtendo-se no final do fluxo de compilação um módulo monolítico, contendo o código máquina e os dados quer da aplicação quer do executivo.

Na realidade são disponibilizadas três bibliotecas, duas correspondendo às implementações no processador *ARPA-MT* descrito no apêndice A e uma para as implementações nas plataformas *MB-SoC* e *PPC-SoC*, descrita no apêndice B. No primeiro caso as duas implementações distinguem-se pelo suporte ou não do coprocessador *Cop2-OSC*, designadas por *OReK-ARPA-C2* e *OReK-ARPA-Sw*, respectivamente. Por último, na implementação do executivo *OReK* nos processadores *MicroBlaze* e *PowerPC 405*, designados por *OReK-MB* e *OReK-PPC*, respectivamente, todas as suas funcionalidades são executadas completamente em *software*.

Todas as bibliotecas implementam a mesma interface. Desta forma é possível utilizar o mesmo código fonte da aplicação independentemente da plataforma.

3.1.2 Características Gerais

No estado actual, o executivo *OReK* fornece serviços de temporização, gestão de tarefas e sincronização no acesso a recursos partilhados com base em semáforos binários e primitivas de controlo de preempção das tarefas. Na implementação destinada ao processador *ARPA-MT* é também capaz de gerir os diversos contextos de execução, podendo controlar as capacidades de multi-tarefa simultânea do processador.

As resoluções temporais permitidas dependem da plataforma base e da dimensão do conjunto de tarefas, o qual define os requisitos computacionais das funções internas do executivo. Com o suporte do coprocessador *Cop2-OSC* podem ser alcançados valores tão pequenos quanto um micro segundo.

Em termos genéricos, os serviços disponibilizados pelo executivo *OReK* dividem-se nos seguintes grupos:

- Configuração e estado (arranque, terminação e informação temporal);
- Diagnóstico (códigos de erro e mensagens);

- Gestão de tarefas (criação, destruição, manipulação, etc.);
- Gestão de semáforos (criação, destruição, manipulação, etc.).

Os tipos de tarefas considerados na implementação do executivo *OReK* foram os seguintes:

- Ordinárias (*non real-time*);
- Periódicas de tempo-real não críticas (*soft real-time periodic*) ou simplesmente periódicas;
- Aperiódicas de tempo-real não críticas (*soft real-time aperiodic*) ou simplesmente aperiódicas;
- Periódicas de tempo-real críticas (*hard real-time periodic*) ou simplesmente periódicas de tempo-real;
- Aperiódicas de tempo-real críticas (*hard real-time aperiodic*) ou simplesmente aperiódicas de tempo-real.

A tabela 3.1 apresenta os parâmetros suportados pelas tarefas do executivo *OReK*. Além destes, existe ainda a possibilidade de especificar o contexto de execução de cada tarefa. Porém, apenas o processador *ARPA-MT* suporta vários contextos de execução [Arn07], pelo que, para as plataformas *MB-SoC* e *PPC-SoC*, todas as tarefas terão o mesmo contexto de execução.

Tarefas	Prioridade base	Execução única	Identificação do grupo	Tamanho da pilha	Período	Fase inicial	*MIT*	Ligação a interrupção externa	*Deadline* relativa
Ordinárias	✓	✓	✓	✓					
Periódicas			✓	✓	✓	✓			
Aperiódicas			✓	✓			✓	✓	
Periódicas de tempo-real			✓	✓	✓	✓			✓
Aperiódicas de tempo-real			✓	✓			✓	✓	✓

Tabela 3.1: Quadro resumo dos parâmetros suportados pelas tarefas do executivo *OReK*.

Dentro de cada contexto de execução, o escalonamento das tarefas é feito segundo os critérios *EDF* (*Earliest Deadline First*), *RM* (*Rate Monotonic*) e *FIFO* (*First Come-First Served*) para as tarefas de tempo-real críticas, tempo-real não críticas e ordinárias, respectivamente.

Os semáforos implementados no executivo *OReK* são do tipo binário, usados para garantir a exclusão mútua na entrada em regiões críticas onde é feito o acesso a recursos partilhados.

Para gerir os semáforos é usado o protocolo *SRP* (*Stack Resource Policy*), uma vez que pode ser usado em conjunto com políticas de escalonamento baseadas em prioridades estáticas (tais como a *RM* e a *FIFO*) ou dinâmicas (tal como a *EDF*). Além disso, tal como mencionado no capítulo 2, o protocolo *SRP* limita o tempo de bloqueio de tarefas de alta prioridade

devido a semáforos detidos por tarefas de menor prioridade e previne a ocorrência de bloqueios em cadeia e de *deadlocks*.

Estas funcionalidades estão implementadas completamente em *software* nas versões *OReK-ARPA-Sw*, *OReK-MB* e *OReK-PPC*, e de forma híbrida em *hardware-software* na versão *OReK-ARPA-C2* com base nos serviços disponibilizados pelo *Cop2-OSC*.

No caso das implementações completas em *software*, além das funções que implementam os serviços disponibilizados à aplicação, o executivo inclui ainda as funções `TimerCallback` e `IntCallback`. A primeira deve ser invocada no contexto da rotina de serviço à interrupção do temporizador do sistema e a sua função é executar o processamento periódico, o escalonamento e a comutação da tarefa em execução. A segunda é invocada a partir da rotina de serviço às interrupções dos periféricos para activação das tarefas aperiódicas correspondentes.

Por outro lado, na implementação suportada pelo coprocessador *Cop2-OSC* é disponibilizada a rotina `OSCCallback` cuja função é apenas a comutação da tarefa em execução, devendo para tal ser invocada no contexto da rotina de tratamento das excepções geradas pelo coprocessador *Cop2-OSC*.

Embora com diferentes desempenhos, todas as implementações disponibilizam as mesmas funcionalidades e a mesma interface do ponto de vista da aplicação o que é vantajoso no panorama do desenvolvimento de aplicações e possibilita a comparação directa entre as implementações integrais em *software* do executivo *OReK* e uma em que as primitivas básicas são realizadas em *hardware*.

3.2 Utilização do Paradigma de Orientação por Objectos

O executivo *OReK* foi desenvolvido com a linguagem *C++*, a qual suporta o Paradigma de Orientação por Objectos (*Object Oriented Paradigm - OOP*). Este paradigma no desenvolvimento de *software* consiste na criação de aplicações constituídas por vários objectos que interagem. Cada objecto é uma estrutura de dados que integra também um conjunto de funções internas que operam sobre os campos de dados e um conjunto de funções de interface que permitem trocar informação com o exterior. A utilização do *OOP* no desenvolvimento do executivo e na implementação das tarefas foi motivada por algumas potencialidades interessantes deste paradigma, nomeadamente, a abstracção de dados, a herança, o encapsulamento e o polimorfismo. Em conjunto, estes mecanismos permitem a construção de programas mais concisos e legíveis e, facilitam a reutilização de código pré-existente. Estas facilidades podem ser descritas sumariamente da seguinte forma:

- Abstracção de dados - uma linguagem baseada no *OOP* permite definir novos tipos de dados (representados por classes na linguagem *C++*) e as operações que sobre eles podem ser executadas (métodos e operadores em *C++*). Esta funcionalidade, em conjunto com a capacidade de efectuar a sobrecarga (redefinição) das funções e dos operadores, permite utilizar intuitivamente os novos tipos de dados definidos pelo utilizador de forma análoga aos predefinidos na linguagem. Uma vez definida uma classe, podem-se declarar objectos dessa classe, isto é, criar instâncias da classe, da mesma forma que se declaram variáveis dos tipos predefinidos da linguagem.

- Herança - a partir de uma classe base podem ser derivadas uma ou mais classes. Através do mecanismo de herança é possível modificar e/ou estender, numa classe derivada, a funcionalidade da classe base. A classe derivada pode possuir novos atributos e métodos, bem como estender e/ou redefinir os métodos da classe base, facilitando assim a reutilização de código pré-existente. O desenvolvimento é também simplificado, uma vez que tudo o que é comum a um conjunto de classes pode ser colocado numa ou várias classes base.

- Encapsulamento - no caso do *C++*, uma classe possui variáveis e funções, também vulgarmente designadas por atributos e métodos, respectivamente. A visibilidade ou acessibilidade dos atributos e métodos pode ser individualmente especificada. Existem três níveis de acessibilidade distintos: privado, protegido e público. Por omissão os elementos são privados, isto é, são acessíveis apenas dentro da classe em que estão definidos. Os atributos e métodos protegidos são também acessíveis nas classes derivadas. Finalmente, um elemento público é visível em qualquer parte do programa onde o objecto possa ser acedido. O encapsulamento consiste geralmente em tornar privados ou protegidos os atributos da classe e públicos os seus métodos de interface. Isto significa que o acesso aos atributos é feito a partir dos métodos da classe, o que facilita a manutenção da integridade interna do objecto.

- Polimorfismo - este mecanismo permite que o método invocado para um dado objecto seja determinado durante a execução do programa em função do tipo efectivo do objecto, em vez de ser determinado estaticamente pelo compilador. No *C++* o polimorfismo é implementado através de funções virtuais e de ponteiros/referências para objectos.

No executivo *OReK*, uma tarefa é um objecto implementado em *C++* através de uma classe derivada da `COReKTask`. Esta fornece um conjunto de serviços base para manipulação de tarefas, tais como criação, destruição, terminação, arranque, paragem, activação, etc. Tal como qualquer classe, uma tarefa possui pelo menos um constructor e método(s), podendo também ter um destructor e atributos. Uma classe que implementa uma tarefa pode ser instanciada diversas vezes numa aplicação correspondendo cada uma a uma tarefa distinta que executa concorrentemente com as outras tarefas do sistema. Cada instância é tipicamente configurada durante a sua criação através de parâmetros do respectivo constructor ou método de inicialização. Uma tarefa pode também possuir vários métodos específicos da aplicação. Contudo, existe um mandatório e que define o ponto de entrada da tarefa, isto é, o método invocado pelo executivo quando é iniciada a execução da tarefa. A sua assinatura, ou protótipo, deve ser o seguinte:

```
virtual void Main();
```

A classe `COReKTask` não é instanciável porque é abstracta, uma vez que o método de entrada da tarefa está definido como uma função virtual pura, isto é, não possui implementação.

Em suma, na implementação do executivo *OReK* a herança é utilizada para colocar numa classe base abstracta todas as estruturas de dados e de controlo comuns a todas as tarefas. O encapsulamento permite esconder essas estruturas. O polimorfismo simplifica a implementação do método de entrada da tarefa.

3.2.1 Tipos de Variáveis

Do ponto de vista da duração, num programa existem em geral três tipos de variáveis, correspondendo cada um a diferentes zonas ou segmentos de dados:

- Automáticas - declaradas no corpo das funções ou métodos. As variáveis automáticas são colocadas em registos internos do processador ou na pilha, criadas aquando da sua declaração e destruídas automaticamente quando o bloco onde foram declaradas termina.

- Estáticas - estas variáveis podem ser declaradas dentro ou fora das funções ou métodos, são colocadas no segmento de dados estáticos, criadas e inicializadas quando o programa é carregado em memória ou inicia a sua execução e destruídas automaticamente quando o programa termina.

- Dinâmicas - as variáveis deste tipo são acedidas através de ponteiros para uma zona de dados específica designada por *heap*. As variáveis dinâmicas são criadas e destruídas explicitamente durante a execução do programa usando primitivas para alocação e libertação de memória tais como as funções `malloc` e `free` do *C* ou os operadores `new` e `delete` do *C++*.

As linguagens baseadas no OOP possuem ainda outro tipo de variável já mencionado acima, os atributos.

No contexto das tarefas, as variáveis automáticas são específicas de cada invocação de uma tarefa, isto é, não são preservadas entre diferentes instâncias da mesma tarefa. As variáveis estáticas são preservadas durante toda a execução do programa e partilhadas por todas as tarefas. Os atributos da tarefa são específicos de cada tarefa e preservados entre diferentes activações da mesma tarefa. Isto permite declarar facilmente variáveis que são específicas de cada tarefa e preservadas entre activações, sem a necessidade de passar explicitamente parâmetros à função que implementa a tarefa.

3.3 Arquitectura

A figura 3.1 ilustra a constituição típica de um sistema de tempo-real baseado no executivo *OReK*. O executivo cria uma camada de abstracção que esconde alguns detalhes do *hardware*, em particular os relativos à gestão dos temporizadores e interrupções e disponibiliza serviços de temporização, de gestão de tarefas e manipulação de semáforos.

Uma aplicação baseada no executivo *OReK* é monolítica, isto é, consiste num único módulo executável ou imagem binária contendo quer a implementação das tarefas da aplicação quer o executivo *OReK*.

O sistema é composto por *M* tarefas, uma de entrada (índice 0) e *M-1* da aplicação. As tarefas da aplicação são executadas de acordo com a sua prontidão e prioridade. A tarefa de entrada é executada no arranque do sistema e quando não houver qualquer outra tarefa para executar.

Cada tarefa possui código e dados. O código é partilhado por todas as instâncias da mesma classe de tarefas, enquanto os dados (atributos e variáveis automáticas) são específicos de cada

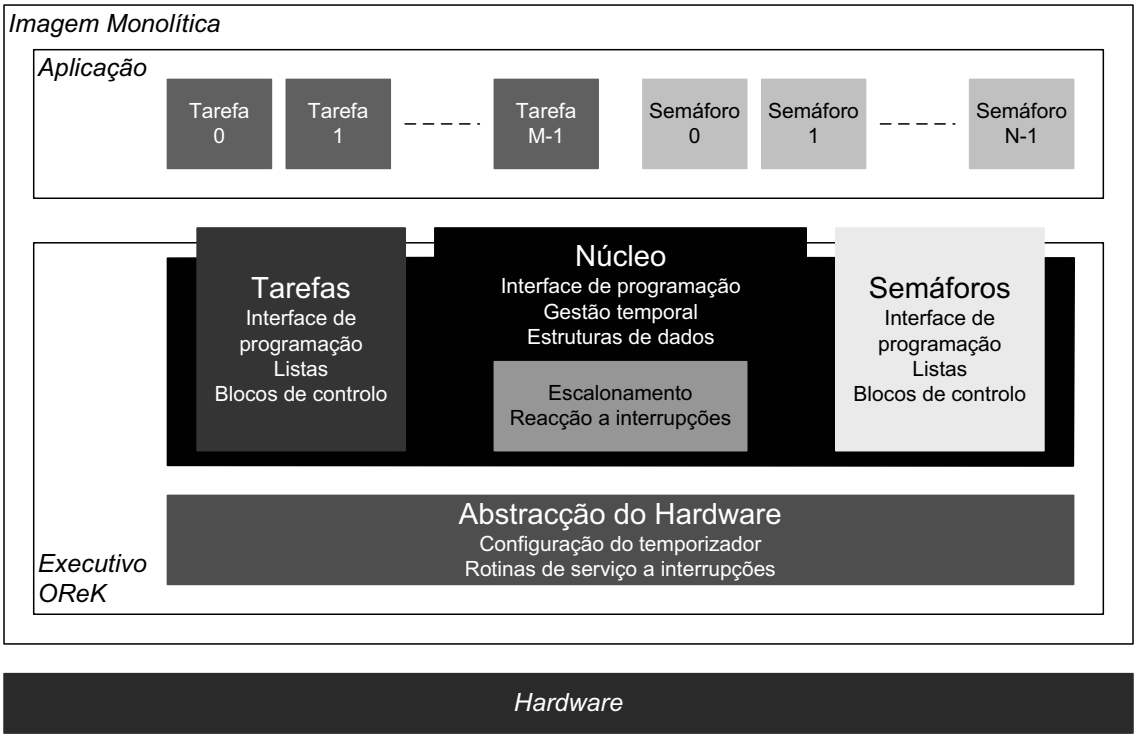


Figura 3.1: Arquitectura de uma aplicação baseada no executivo *OReK* (retirado de [Arn07]).

tarefa.

O executivo *OReK* é preemptivo, isto é, permite que a execução de uma tarefa seja interrompida em qualquer instante sem que para tal a própria tarefa tenha que se auto-suspender explicitamente. Independentemente do ponto onde a tarefa se encontre, o executivo é capaz de interromper a sua execução, comutar para outra(s) tarefa(s) e mais tarde retomar a execução da primeira tarefa a partir do ponto onde foi interrompida.

O contexto de uma tarefa é composto por:

- Conteúdo do registo *Program Counter - PC* (par de registos *CS:IP* na arquitectura *Intel x86*) ou registo *PC* dos processadores *ARPA-MT* e *MicroBlaze*. De referir que a arquitectura do processador *PowerPC 405* não disponibiliza nenhum registo para acesso do *Program Counter*;
- Conteúdo do registo de estado, se aplicável (*FLAGS* na arquitectura *Intel x86* e registo *Machine Status Register - MSR* nos processadores *MicroBlaze* e *PowerPC 405*);
- Conteúdo de todos os registos internos do processador utilizáveis pelas tarefas (registos de uso geral, registos de coprocessadores, etc.);
- Pilha (*stack*) e respectivo ponteiro.

Além das tarefas, o sistema é constituído por N semáforos. O primeiro (índice 0) é dedicado ao controlo de acesso aos coprocessadores *Cop0-MEC* e *Cop2-OSC* do processador

ARPA-MT. Os restantes *N-1* semáforos estão disponíveis para fins genéricos dependentes da aplicação. No entanto, ao contrário do que acontece com as tarefas, durante a configuração do executivo, o número máximo de semáforos pode ser especificado como sendo zero.

Uma vez que o índice 0 quer das tarefas, quer dos semáforos é usado para fins específicos, o valor zero é também usado com o significado de identificador inválido.

3.3.1 Núcleo

O componente central do executivo *OReK* é o seu núcleo. Internamente, o núcleo responde às interrupções do temporizador e dos periféricos, executa o algoritmo de escalonamento e efectua a comutação das tarefas. Disponibiliza à aplicação os seguintes serviços:

- Inicialização e terminação do núcleo;
- Criação e destruição de tarefas;
- Arranque e paragem de tarefas;
- Informação temporal e de estado das tarefas;
- Gestão de grupos de tarefas;
- Criação, destruição e gestão de semáforos;
- Controlo da preempção das tarefas.

O temporizador de *hardware* gera interrupções periodicamente. Sempre que ocorre uma interrupção é executado o algoritmo de escalonamento das tarefas e eventualmente feita a comutação de tarefas. Tudo isto é feito no âmbito da rotina de serviço à interrupção do temporizador. No início desta rotina é necessário salvaguardar o contexto da tarefa actual, seguidamente actualizar os parâmetros temporais dinâmicos das tarefas, determinar a próxima tarefa a executar e no final da rotina restaurar o contexto da próxima tarefa que vai ser executada.

No caso de no contexto da aplicação ser executada uma operação que faça com que a tarefa activa transite para um estado não activo (e.g. terminação da tarefa) é também gerada por *software* uma pseudo-interrupção do temporizador de forma a que seja invocada a respectiva rotina de serviço onde é feito o escalonamento e a comutação de tarefas.

O conteúdo do *PC* é automaticamente salvaguardado quando ocorre uma interrupção. O registo de estado é automaticamente salvaguardado por *hardware* na arquitectura *Intel x86*. O mesmo não acontece nos processadores *MicroBlaze* e *PowerPC 405* onde a salvaguarda do registo de estado tem de ser feita explicitamente por *software*. Os restantes registos também têm de ser explicitamente salvaguardados pelo executivo.

A troca da pilha activa é feita através da alteração do valor do registo usado como ponteiro da pilha de forma a apontar para o topo da pilha da próxima tarefa que vai ser executada.

3.3.2 Tarefas

As tarefas são objectos que encapsulam o código e os dados e implementam a funcionalidade da aplicação. No executivo *OReK*, uma tarefa pode estar num dos seguintes estados:

- *Free* - (Livre);
- *Stopped* - (Parada);
- *Idle* - (Inactiva);
- *Ready* - (Pronta);
- *Running* - (A executar);
- *Preempted* - (Interrompida).

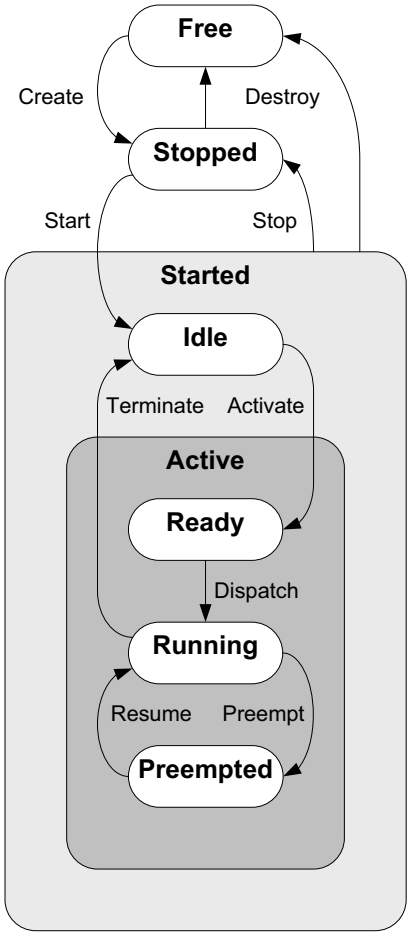


Figura 3.2: Estados das tarefas e possíveis transições no executivo *OReK* (retirado de [Arn07]).

A figura 3.2 ilustra os vários estados em que uma tarefa se pode encontrar, bem como as transições permitidas.

Qualquer tarefa que se encontre num dos estados *Idle*, *Ready*, *Running* ou *Preempted* diz-se iniciada (*Started*), caso contrário diz-se parada (*Stopped*). Uma tarefa que se encontre num dos estados *Ready*, *Running* ou *Preempted* diz-se activa (*Active*).

Uma tarefa quando é criada é colocada no estado *Stopped*, permanecendo nesse estado até ser explicitamente iniciada. Depois de iniciada é colocada no estado *Idle* até ser alcançado o instante da primeira activação. Nessa altura transita para o estado *Ready*. Uma tarefa no estado *Ready* está pronta a executar. A passagem para o estado *Running* é feita pelo algoritmo de escalonamento através da atribuição de tempo de processador à tarefa. Uma tarefa a executar pode ser interrompida de forma a ser executada outra mais prioritária, passando nesta situação para o estado *Preempted*. Uma tarefa depois de terminar é destruída ou colocada no estado *Idle* até à próxima activação ou instância.

Em qualquer estado pode ser dada ordem de paragem ou de destruição de uma tarefa, transitando neste caso para o estado *Stopped* ou *Free*, respectivamente. A transição de estado pode ser deferida ou imediata, consoante a tarefa já tenha iniciado ou não a sua execução, respectivamente. Uma tarefa depois de destruída, deixa de existir no sistema.

Os estados *Sleeping*, *Suspended* e *Blocked* normalmente implementados nos executivos ou sistemas operativos de tempo-real, não foram incluídos no executivo *OReK* pelos motivos expostos no capítulo 2, relacionados com a implementação do protocolo *SRP*.

3.3.3 Semáforos

Os semáforos são objectos que encapsulam os mecanismos de sincronização do executivo *OReK*. Um semáforo pode estar num dos seguintes estados:

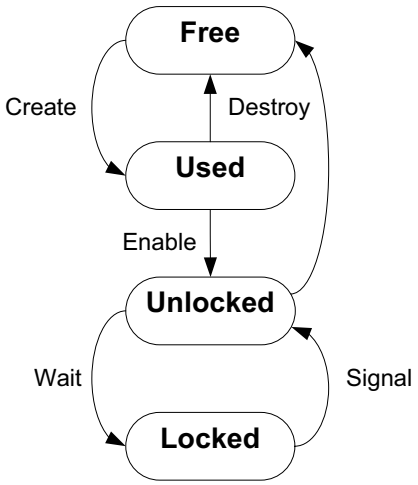


Figura 3.3: Estados dos semáforos e possíveis transições no executivo *OReK* (retirado de [Arn07]).

- *Free* - (Livre);
- *Used* - (Usado);
- *Unlocked* - (Desbloqueado);
- *Locked* - (Bloqueado).

A figura 3.3 ilustra os vários estados em que um semáforo se pode encontrar, bem como as transições permitidas.

Um semáforo quando é criado é colocado no estado *Used*. Neste estado deve ser inicializado o respectivo tecto do recurso. Seguidamente pode ser activado, transitando para o estado *Unlocked*. Um semáforo transita do estado *Unlocked* para o estado *Locked* através da execução da primitiva *Wait*. O retorno ao estado *Unlocked* faz-se por intermédio da primitiva *Signal*. Um semáforo pode ser destruído de não estiver no estado *Locked*. Um semáforo depois de destruído, deixa de existir no sistema.

3.4 Implementação

As próximas subsecções descrevem alguns detalhes relativos à implementação do executivo *OReK*, nomeadamente, as linguagens de programação, as classes e estruturas, a organização dos ficheiros e a realização da gestão de tarefas e da sua sincronização com base em semáforos.

3.4.1 Linguagens e Independência da Plataforma

A generalidade do executivo *OReK* é independente da plataforma alvo e está escrito em *C++* portável. A linguagem *C++* foi a escolhida por ser:

- Eficiente;
- Fácil de realizar a interface com módulos escritos noutras linguagens, nomeadamente em *C* e *assembly*;
- Orientada por objectos;
- Popular e bem suportada por ferramentas para diversas plataformas.

Algumas partes de baixo nível, mas independentes da plataforma estão implementadas em linguagem *C*, igualmente portável.

Existem ainda pequenos segmentos de código que são específicos da plataforma, tais como as funções de programação do temporizador, o atendimento de baixo nível das interrupções dos periféricos, as funções onde se acede explicitamente aos registos do processador e os segmentos da rotina de serviço à interrupção do temporizador onde é feita a comutação do contexto da tarefa. Nestes casos foi utilizada a linguagem *assembly* do processador alvo.

Apesar de no estado actual o executivo *OReK* ser suportado apenas nas plataformas *Intel x86/MSDOS*, *ARPA-MT*, *MB-SoC* e *PPC-SoC*, houve o cuidado de isolar todos os blocos de código que sejam dependentes da plataforma. Desta forma é relativamente simples portar o executivo para outras plataformas, bastando para tal reescrever esses blocos.

Um aspecto específico da arquitectura é a pilha da tarefa. De forma a poder ser feita a comutação das tarefas, sempre que ocorre uma interrupção do temporizador, o contexto da tarefa interrompida deve ser armazenado na pilha da tarefa, sendo o respectivo ponteiro guardado no bloco de controlo dessa tarefa. Essa informação será usada posteriormente quando for retomada a execução da tarefa interrompida.

3.4.2 Classes e Estruturas

A figura 3.4 ilustra as componentes, isto é, os módulos e as classes principais do executivo *OReK* que implementam a arquitectura representada na figura 3.1.

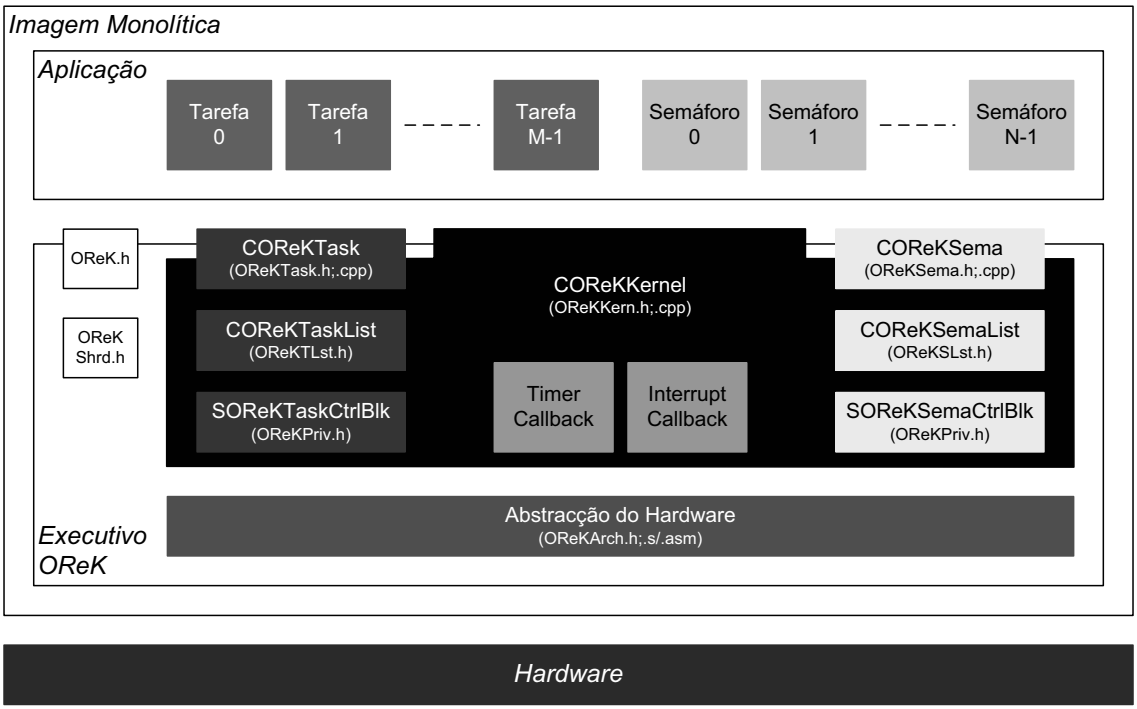


Figura 3.4: Estrutura da implementação em software do executivo *OReK* (retirado de [Arn07]).

Mais concretamente são mostradas as classes `COReKKern`, `COReKTask`, `COReKTaskList`, `COReKSema` e `COReKSemaList` e as estruturas de dados `SOReKTaskCtrlBlk` e `SOReKSemaCtrlBlk`. São também mostradas na figura 3.4 as funções executadas no contexto das rotinas de serviço às interrupções do temporizador e dos periféricos (*timer callback* e *interrupt callback*). As subsecções seguintes descrevem sucintamente cada um destes elementos.

3.4.2.1 A Classe COReKKern

A classe `COReKKern` implementa o núcleo do executivo *OReK*. As figuras 3.5 e 3.6 mostram, respectivamente, a interface (métodos públicos) e os atributos protegidos desta classe. Em conjunto implementam os serviços disponibilizados pelo núcleo e enumerados acima.

```
class COReKKern
{
// Methods
public:
    static int Initialize(uint maxNumTasks, uint maxNumSemas,
                          uint baseFrequency, uint timeResolution
    #ifndef __ARPA__
        #if !defined (__MICROBLAZE__) && !defined (__PPC__)
                         , ostream* pLogStream = NULL
        #endif // __MICROBLAZE__  && __PPC__
    #endif // __ARPA__
                          );
    static int ShutDown(void);
    static uhuge GetTickCount(void);
    static void DisablePreemption(void);
    static void EnablePreemption(void);

    static int CreateNonRTTask(COReKTask* pTask, handle* phTask,
                               uchar basePrio, bool singlShot,
                               uchar taskGrpId = 0,
                               uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                               uchar runContxt = 0);

    static int CreateSoftRTPerTask(COReKTask* pTask, handle* phTask,
                                   int period, int initPhase,
                                   uchar taskGrpId = 0,
                                   uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                                   uchar runContxt = 0);

    static int CreateSoftRTApeTask(COReKTask* pTask, handle* phTask,
                                   int minITime, int intBind,
                                   uchar taskGrpId = 0,
                                   uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                                   uchar runContxt = 0);

    static int CreateHardRTPerTask(COReKTask* pTask, handle* phTask,
                                   int period, int relDdlin, int initPhase,
                                   uchar taskGrpId = 0,
                                   uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                                   uchar runContxt = 0);

    static int CreateHardRTApeTask(COReKTask* pTask, handle* phTask,
                                   int minITime, int relDdlin, int intBind,
                                   uchar taskGrpId = 0,
                                   uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                                   uchar runContxt = 0);

    static int DestroyTask(handle hTask);
    static int StartTask(handle hTask, int initPhase);
    static int StopTask(handle hTask);
    static int StartTaskGroup(uchar taskGrpId);
    static int StopTaskGroup(uchar taskGrpId);
    static int StartAllTasks(void);
    static int StopAllTasks(void);

    static int ActivateTask(handle hTask, int delay);
    static int GetTaskState(handle hTask, TOReKTaskState* pTaskState);
    static int CreateSema(COReKSema* pSema, handle* phSema);
    static int DestroySema(handle hSema);
    static int RegTaskOnSema(handle hSema, handle hTask, bool enable);
    static int EnableSema(handle hSema);
    static int Wait(handle hSema);
    static int Signal(handle hSema);

    static const char* GetStatusTypeStr(int statusCode);
    static const char* GetStatusMsgStr(int statusCode);
};
```

Figura 3.5: Interface da classe `COReKKern` do executivo *OReK*.

```
// Attributes
protected:
static bool m_running;
static uint m_baseFrequency;
static uint m_timeResolution;
static uint m_kernAuxStacks[OREK_NUM_CONTXTS][OREK_KERN_AUX_STACK_SZ];

#ifndef  __ARPA_OSC__

    static uhuge m_tickCount;

    static bool m_preemptEnabled[OREK_NUM_CONTXTS];

    static bool m_taskEndException[OREK_NUM_CONTXTS];
    static bool m_signalException[OREK_NUM_CONTXTS];

#endif // __ARPA_OSC__

    static int m_maxNumTasks;
    static TOReKTaskCtrlBlk* m_pTCBPool;

#ifndef  __ARPA_OSC__

    static COReKTaskList m_freeTCBs;

    static COReKTaskList m_stoppedTaskList;
    static COReKTaskList m_idlePerTaskList[OREK_NUM_CONTXTS];
    static COReKTaskList m_idleApeTaskList;
    static COReKTaskList m_readyNonRTTaskList[OREK_NUM_CONTXTS];
    static COReKTaskList m_readySoftRTTaskList[OREK_NUM_CONTXTS];
    static COReKTaskList m_readyHardRTTaskList[OREK_NUM_CONTXTS];
    static COReKTaskList m_preemptedTaskList[OREK_NUM_CONTXTS];

    static TOReKTaskCtrlBlk* m_pIntBindTasks[OREK_NUM_INTERRUPTS];

#endif // __ARPA_OSC__

    static TOReKTaskCtrlBlk* m_pRunningTasks[OREK_NUM_CONTXTS];

    static int m_maxNumSemas;
    static TOReKSemaCtrlBlk* m_pSCBPool;

#ifndef  __ARPA_OSC__

    static COReKSemaList m_freeSCBs;

    static uint* m_pCeilStackBlks;
    static uint* m_pPrevContxtsCeil[OREK_NUM_CONTXTS];
    static uint m_contxtsCeiling[OREK_NUM_CONTXTS];

#endif // __ARPA_OSC__

#ifndef __ARPA__
    #if !defined (__MICROBLAZE__) && !defined (__PPC__)
        static ostream* m_pLogStream;
    #endif // __MICROBLAZE__ && __PPC__
#endif // __ARPA__
```

Figura 3.6: Atributos da classe `COReKKern` do executivo *OReK*.

Os nomes dos métodos e atributos são auto-explicativos pelo que não serão considerados individualmente para não tornar esta discussão enfadonha. Além disso, os métodos relacionados com a gestão de tarefas e de semáforos não se destinam a ser invocados directamente pela aplicação mas indirectamente através das respectivas classes `COReKTask` e `COReKSema`.

Todos os métodos e atributos desta classe são estáticos e a classe não é instanciável porque não faz sentido existir mais do que um núcleo no sistema.

Os atributos listados na figura 3.6 entre as directivas `#ifndef __ARPA_OSC__` e `#endif // __ARPA_OSC__` são incluídos nas implementações *OReK-ARPA-Sw*, *OReK-MB* e *OReK-PPC* uma vez que na versão *OReK-ARPA-C2* correspondem a campos dos registos do coprocessador *Cop2-OSC*.

3.4.2.2 A Classe COReKTask

A classe `COReKTask` fornece uma definição abstracta de uma tarefa, pelo que não pode ser directamente instanciada. Esta classe simplifica a manipulação das tarefas, guardando internamente o identificador ou *handle* da tarefa devolvido pelo núcleo aquando da criação da mesma. Sempre que necessário utiliza esse identificador para invocar a função do núcleo correspondente. A figura 3.7 mostra a interface desta classe e que consiste num construtor, num destrutor e nos seguintes métodos:

- `CreateNonRT` - cria uma tarefa ordinária;
- `CreateSoftRTPer` - cria uma tarefa periódica;
- `CreateSoftRTApe` - cria uma tarefa aperiódica;
- `CreateHardRTPer` - cria uma tarefa periódica de tempo-real;
- `CreateHardRTApe` - cria uma tarefa aperiódica de tempo-real;
- `Destroy` - destrói a tarefa;
- `Start` - arranca a tarefa;
- `Stop` - pára a tarefa;
- `Activate` - activa a tarefa (aplicável apenas a tarefas aperiódicas);
- `GetState` - devolve o estado da tarefa.

Além destes métodos, define ainda a assinatura do método de entrada da tarefa (`virtual void Main() = 0`). Este método não está implementado nesta classe e deve ser definido em todas as classes que implementam tarefas concretas, as quais têm que ser derivadas da classe `COReKTask`.

Esta classe pode invocar os serviços do núcleo do executivo directamente ou através de chamadas ao sistema. A primeira abordagem pode ser empregue quando a aplicação e o executivo são integrados numa única imagem monolítica. A segunda abordagem é usada quando a aplicação e o executivo são carregados como módulos separados ou executam em diferentes modos de operação do processador ou espaços de endereçamento.

```
class COReKTask
{
    friend class COReKSema;

// Constructors / Destructors prototypes
public:
    COReKTask();
    virtual ~COReKTask();

// Methods prototypes
public:
    int CreateNonRT(uchar basePrio, bool singlShot,
                    uchar taskGrpId = 0,
                    uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                    uchar runContxt = 0);

    int CreateSoftRTPer(int period, int initPhase,
                        uchar taskGrpId = 0,
                        uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                        uchar runContxt = 0);

    int CreateSoftRTApe(int minITime, int intBind,
                        uchar taskGrpId = 0,
                        uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                        uchar runContxt = 0);

    int CreateHardRTPer(int period, int relDdlin, int initPhase,
                        uchar taskGrpId = 0,
                        uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                        uchar runContxt = 0);

    int CreateHardRTApe(int minITime, int relDdlin, int intBind,
                        uchar taskGrpId = 0,
                        uint stackSize = OREK_TASK_DEFAULT_STACK_SZ,
                        uchar runContxt = 0);

    int Destroy();
    int Start(int initPhase);
    int Stop();
    int Activate(int delay);

    int GetState(TOReKTaskState* pTaskState);

    virtual void Main() = 0;

// Attributes
protected:
    handle m_handle;
};
```

Figura 3.7: Interface da classe `COReKTask` do executivo *OReK*.

3.4.2.3 A Classe COReKSema

A classe `COReKSema` define as funcionalidades acessíveis à aplicação disponibilizadas pelos semáforos implementados no executivo *OReK*. Esta classe simplifica a manipulação dos semáforos, guardando internamente o identificador ou *handle* do semáforo devolvido pelo núcleo aquando da criação do mesmo. Sempre que necessário utiliza esse identificador para invocar a função do núcleo correspondente.

Ao contrário do que acontecia com as tarefas, esta classe pode ser instanciada directamente. A figura 3.8 ilustra a interface pública desta classe, sendo constituída por um constructor, um destructor e pelos seguintes métodos:

- `Create` - cria um semáforo;
- `Destroy` - destrói o semáforo;
- `Enable` - activa o semáforo;
- `RegisterTask` - regista uma tarefa como utilizadora do recurso controlado pelo semáforo;
- `Wait` - bloqueia o semáforo;
- `Signal` - desbloqueia o semáforo.

```
class COReKSema
{
    friend class COReKTask;

// Constructors / Destructors prototypes
public:
    COReKSema();
    virtual ~COReKSema();

// Methods prototypes
public:
    int Create();
    int Destroy();

    int RegisterTask(const COReKTask& task, bool enable = false);
    int Enable();

    int Wait();
    int Signal();

// Attributes
protected:
    handle m_handle;
};
```

Figura 3.8: Interface da classe `COReKSema` do executivo *OReK*.

Após a construção de um semáforo, todas as tarefas que acedem ao recurso controlado pelo semáforo, devem ser registadas de forma a configurar o respectivo tecto. Este requisito

deve-se ao facto de, apesar das suas importantes vantagens, a utilização do protocolo *SRP* na gestão dos semáforos não ser transparente para a aplicação.

Tal como na classe `COReKTask`, esta classe pode invocar os serviços do núcleo do executivo directamente ou através de chamadas ao sistema.

3.4.2.4 A Estrutura SOReKTaskCtrlBlk

A estrutura `SOReKTaskCtrlBlk` define o bloco de controlo da tarefa (*Task Control Block - TCB*). Esta estrutura possui vários campos onde são armazenados os parâmetros de configuração e guardado o estado da respectiva tarefa. A sua definição possui duas partes (figura 3.9):

- A primeira (antes da directiva `#ifndef __ARPA_OSC__`) - independente da implementação;
- A segunda (depois da directiva `#ifndef __ARPA_OSC__`) - incluída nas versões *OReK-ARPA-Sw* ,*OReK-MB* e *OReK-PPC*, uma vez que estes campos fazem parte do coprocessador *Cop2-OSC* na implementação *OReK-ARPA-C2*.

O campo `m_pTask` é um ponteiro para um objecto ou instância de uma classe derivada da `COReKTask` onde é implementada a funcionalidade da tarefa propriamente dita. É através deste ponteiro que o executivo invoca o método de entrada da tarefa.

O campo `m_stackSize` contém o tamanho da pilha associada à tarefa medido em palavras nativas da arquitectura base (palavra de 16 bits na plataforma *Intel x86/MSDOS* ou de 32 bits na plataforma *ARPA-MT* e nos processadores *MicroBlaze* e *PowerPC 405*). Este valor é estabelecido durante a criação da tarefa não podendo ser alterado posteriormente.

O campo `m_pStackBlk` armazena o endereço inicial do bloco de memória onde reside a pilha da tarefa.

O campo `m_currentSP` armazena o valor do registo da *CPU* que desempenha o papel de *stack pointer* inicial da tarefa ou no instante em que é interrompida, para que possa ser retomada mais tarde. De notar que todo o estado da tarefa é armazenado na respectiva pilha, pelo que é suficiente guardar o endereço do topo no momento da preempção.

Nas implementações *OReK-ARPA-Sw*, *OReK-MB* e *OReK-PPC*, uma tarefa está numa das listas de tarefas internas do núcleo. Cada lista possui um estado associado. O campo `m_pTaskList` é um ponteiro para a lista na qual a tarefa se encontra num dado momento. Os campos `m_pPrevious` e `m_pNext` são ponteiros utilizados pela classe `COReKTaskList` na implementação das listas bi-ligadas de *TCBs*.

Para optimizar simultaneamente o acesso arbitrário a qualquer bloco assim como a inserção ou remoção de blocos, as listas de tarefas são bi-ligadas e implementadas sobre uma tabela (*array*) de blocos de tamanho predefinido alocada estaticamente. Algumas das listas são mantidas ordenadas enquanto outras são manipuladas como filas. Sobre a mesma tabela são implementadas várias listas correspondendo cada uma a um dos estados em que uma tarefa se pode encontrar. Além dessas, existe também uma lista de blocos livres. A utilização de uma tabela tem a vantagem de simplificar a validação do identificador (*handle*) da tarefa, o qual corresponde ao índice do *TCB* dentro da tabela.

```
typedef struct SOReKTaskCtrlBlk
{
    COReKTask* m_pTask;

    uint  m_stackSize;
    uint* m_pStackBlk;

    uint  m_currentSP;

#ifndef  __ARPA_OSC__

    COReKTaskList*    m_pTaskList;
    SOReKTaskCtrlBlk* m_pPrevious;
    SOReKTaskCtrlBlk* m_pNext;

    union
    {
        struct
        {
            uint m_taskType:3;
            uint m_dtryPend:1;
            uint m_stopPend:1;
            uint m_actvPend:1;
            uint m_rsvdBits1:3;
            uint m_ddlinMiss:1;
            uint m_rsvdBits2:2;
            uint m_runContxt:3;
            uint m_singlShot:1;
            uint m_taskGrpId:8;
            uint m_intRqBind:1;
            uint m_intNumber:4;
            uint m_rsvdBits3:3;
        }m_headerBits;
        uint m_headerWord;
    };

    union
    {
        uint m_basePrio;
        uint m_period;
        uint m_minITime;
    };

    union
    {
        uint m_preptLev;
        uint m_relDdlin;
    };

    uint m_activCnt;

    uint m_priority;

#endif // __ARPA_OSC__

}TOReKTaskCtrlBlk;
```

Figura 3.9: Definição da estrutura `SOReKTaskCtrlBlk` do executivo *OReK*.

A utilização de uma tabela estática de tamanho predefinido tem também a vantagem de eliminar o tempo de alocação/libertação de memória necessário numa solução dinâmica, obviamente à custa de uma diminuição da flexibilidade. No entanto, é importante notar que é sempre possível realizar a realocação e redimensionamento da tabela de *TCBs*, embora essa seja uma operação que obriga a suspender temporariamente a execução das tarefas.

O atributo `m_headerWord/m_headerBits` é uma *union* que implementa em *software* um registo com funcionalidade análoga ao campo *THEADER* do *TCB* implementado no coprocessador *Cop2-OSC*, armazenando informação sobre o tipo da tarefa, o seu estado actual, o contexto onde executa, o grupo a que pertence, a associação a fontes de interrupção (possível nas tarefas aperiódicas), além de outros indicadores.

O atributo `m_basePrio/m_period/m_minITime` é uma *union* que implementa em *software* um registo com funcionalidade análoga ao campo *TPERIOD* do *TCB* implementado no coprocessador *Cop2-OSC*, armazenando a prioridade base, o período ou o intervalo mínimo entre activações consoante se trate de uma tarefa ordinária, periódica ou aperiódica, respectivamente.

O atributo `m_preptLev/m_relDdlin` é uma *union* que implementa em *software* um registo com funcionalidade análoga ao campo *TPREPLV* do *TCB* implementado no coprocessador *Cop2-OSC*, armazenando o nível de preempção ou a *deadline* relativa das tarefas.

O atributo `m_activCnt` implementa em *software* um registo com funcionalidade análoga ao campo *TACTIVC* do *TCB* implementado no coprocessador *Cop2-OSC*, armazenando a fase inicial das tarefas periódicas ou o atraso de activação das tarefas aperiódicas. Além disso, é também usado como contador para gerir as reactivações das tarefas periódicas e o intervalo mínimo entre activações das tarefas aperiódicas.

O atributo `m_priority` implementa em *software* um registo com funcionalidade análoga ao campo *TPRIORI* do *TCB* implementado no coprocessador *Cop2-OSC*, armazenando a prioridade actual da tarefa.

3.4.2.5 A Classe COReKTaskList

A classe `COReKTaskList` implementa uma lista bi-ligada de estruturas de dados do tipo `SOReKTaskCTRLBlk` sobre uma tabela predefinida de tamanho fixo. Esta classe disponibiliza os serviços tradicionais para inserção e remoção de elementos da lista além de outras funcionalidades úteis para o escalonamento de tarefas. Para melhorar o desempenho, esta classe utiliza as capacidades *inline* do *C++*.

3.4.2.6 A Estrutura SCOReKSemaCtrlBlk

A estrutura `SOReKSemaCtrlBlk` define o bloco de controlo do semáforo (*Semaphore Control Block - SCB*). Esta estrutura possui vários campos onde são armazenados os parâmetros de configuração e guardado o estado do respectivo semáforo. A sua definição possui duas partes (ver figura 3.10):

- A primeira (antes da directiva `#ifndef __ARPA_OSC__`) - independente da implementação;

- A segunda (depois da directiva `#ifndef __ARPA_OSC__`) - incluída nas versões *OReK-ARPA-Sw* ,*OReK-MB* e *OReK-PPC*, uma vez que estes campos fazem parte do coprocessador *Cop2-OSC* na implementação *OReK-C2*.

```
typedef struct SOReKSemaCtrlBlk
{
    COReKSema* m_pSema;

#ifndef  __ARPA_OSC__

    COReKSemaList* m_pSemaList;
    SOReKSemaCtrlBlk* m_pNext;

    union
    {
        struct
        {
            uchar m_used:1;
            uchar m_enable:1;
            uchar m_locked:1;
        }m_headerBits;
        uchar m_headerWord;
    };

    uint m_resourceCeil;

#endif // __ARPA_OSC__

} TOReKSemaCtrlBlk;
```

Figura 3.10: Definição da estrutura `SOReKSemaCtrlBlk` do executivo *OReK*.

O campo `m_pSema` é um ponteiro para um objecto ou instância da classe `COReKSema` usada para representar o semáforo do lado da aplicação.

Nas implementações *OReK-ARPA-Sw* ,*OReK-MB* e *OReK-PPC*, os *SCBs* livres são guardados numa lista simplesmente ligada. O campo `m_pSemaList` é um ponteiro para a lista na qual o semáforo se encontra num dado momento. O campo `m_pNext` é um ponteiro utilizado pela classe `COReKSemaList` na implementação das listas de *SCBs*.

Tal como acontecia com as tarefas, nos semáforos para optimizar simultaneamente o acesso arbitrário a qualquer bloco de controlo assim como a inserção ou remoção de blocos, as listas de semáforos estão implementadas sobre uma tabela (*array*) de blocos de tamanho predefinido alocada estaticamente.

Os indicadores `m_used`, `m_enable` e `m_locked` indicam o estado do semáforo, possuindo o significado alocado, activo (desbloqueado) e bloqueado, respectivamente.

Finalmente, o campo `m_resourceCeil` armazena o tecto do recurso controlado pelo semáforo, sendo definido como o maior nível de preempção de todas as tarefas que acedem ao recurso controlado em regime de exclusão mútua.

3.4.2.7 A Classe COReKSemaList

A classe `COReKSemaList` implementa uma lista ligada de estruturas de dados do tipo `SOReKSemaCTRLBlk` sobre uma tabela predefinida de tamanho fixo. Esta classe disponibiliza os serviços tradicionais para inserção e remoção de elementos da lista. Para melhorar o desempenho, esta classe utiliza as capacidades *inline* do *C++*.

3.4.3 Outros Tipos e Constantes do Executivo OReK

No sentido de uniformizar as definições do executivo *OReK*, foram desenvolvidos os ficheiros `OReKShrd.h` e `OReKPriv.h` que contêm as definições partilhadas e privadas do executivo *OReK*, respectivamente.

3.4.3.1 Definições Públicas ou Partilhadas com a Aplicação

```
 // Parameters
#define OREK_NUM_CONTXTS             1
#define OREK_NUM_INTERRUPTS          16

#if defined (__MICROBLAZE__) || defined (__PPC__)

    #include <xbasic_types.h>

    #define uchar                    Xuint8
    #define ushort                   Xuint16
    #define ulong                    Xuint32

    typedef unsigned int             uint;
    typedef long long                huge;
    typedef unsigned long long       uhuge;

#else

    typedef unsigned char            uchar;
    typedef unsigned short           ushort;
    typedef unsigned long            ulong;
    typedef unsigned int             uint;

    #ifdef _MSC_VER

        typedef _int64               huge;
        typedef unsigned _int64      uhuge;

    #else

        typedef long long            huge;
        typedef unsigned long long   uhuge;

    #endif // _WIN32

#endif // __MICROBLAZE__ || __PPC__
```

Figura 3.11: Parâmetros e definições de tipos de dados do executivo *OReK*.

No ficheiro de definições partilhadas do executivo *OReK* (`OReKShrd.h`) constam como parâmetros o número de contextos e número de interrupções do executivo, são definidos nomes concretos para os tipos de dados utilizados no executivo *OReK*, de forma a que todas as definições de tipos de dados no executivo sejam independentes da plataforma. São ainda definidos os códigos de estado do executivo *OReK* e definidas as enumerações `EOReKTaskType`, `EOReKTaskState` e `EOReKSemaState`. Estas definem os tipos e estados das tarefas e estados permitidos para os semáforos, respectivamente. As figuras 3.11, 3.12 e 3.13 ilustram as definições partilhadas do executivo *OReK*.

```
#define OREK_INVALID_HANDLE_VALUE               -1

#define OREK_INFO_SUCCESS                       0x00

#define OREK_WARNING_TASK_DESTROY_PENDING       0x01
#define OREK_WARNING_TASK_STOP_PENDING          0x02

#define OREK_ERROR_UNKNOWN_ERROR                0x10
#define OREK_ERROR_OUT_OF_MEMORY                0x11
#define OREK_ERROR_INVALID_PARAMETER            0x12
#define OREK_ERROR_INVALID_HANDLE               0x13

#define OREK_ERROR_KERNEL_ALREADY_RUNNING       0x18
#define OREK_ERROR_KERNEL_NOT_INITIALIZED       0x19
#define OREK_ERROR_CANNOT_SHUT_DOWN_KERNEL      0x1A
#define OREK_ERROR_DEADLINE_MISSED              0x1B

#define OREK_ERROR_TOO_MANY_TASKS               0x20
#define OREK_ERROR_CANNOT_CREATE_TASK           0x21
#define OREK_ERROR_CANNOT_DESTROY_TASK          0x22
#define OREK_ERROR_CANNOT_STOP_TASK             0x23
#define OREK_ERROR_TASK_ALREADY_CREATED         0x26
#define OREK_ERROR_TASK_ALREADY_STARTED         0x27
#define OREK_ERROR_TASK_ALREADY_STOPPED         0x28
#define OREK_ERROR_TASK_NOT_IDLE                0x29

#define OREK_ERROR_SEMAS_NOT_AVAILABLE          0x30
#define OREK_ERROR_TOO_MANY_SEMAS               0x31
#define OREK_ERROR_CANNOT_CREATE_SEMA           0x32
#define OREK_ERROR_CANNOT_DESTROY_SEMA          0x33
#define OREK_ERROR_SEMA_ALREADY_CREATED         0x34
#define OREK_ERROR_SEMA_ALREADY_LOCKED          0x35
#define OREK_ERROR_SEMA_ALREADY_UNLOCKED        0x36
#define OREK_ERROR_SEMA_NOT_ENABLED             0x37

#define OREK_TASK_DEFAULT_STACK_SZ              256u
#define OREK_NRT_PRIO_IDLE                      0x7
#define OREK_NRT_PRIO_LOW                       0x6
#define OREK_NRT_PRIO_BELOW_NORMAL              0x5
#define OREK_NRT_PRIO_NORMAL                    0x4
#define OREK_NRT_PRIO_ABOVE_NORMAL              0x3
#define OREK_NRT_PRIO_HIGH                      0x2
#define OREK_NRT_PRIO_NEAR_RT                   0x1
```

Figura 3.12: Definição dos códigos de estado do executivo *OReK*.

```
typedef enum EOReKTaskType
{
    TT_UNUSED       = 0,
    TT_NON_RT       = 1,
    TT_SOFT_RT_PER  = 2,
    TT_SOFT_RT_APE  = 3,
    TT_HARD_RT_PER  = 4,
    TT_HARD_RT_APE  = 5
}TOReKTaskType;

typedef enum EOReKTaskState
{
    TS_FREE         = -1,
    TS_STOPPED      = 0,
    TS_IDLE         = 1,
    TS_READY        = 2,
    TS_RUNNING      = 3,
    TS_PREEMPTED    = 4
}TOReKTaskState;

typedef enum EOReKSemaState
{
    SS_FREE         = 0,
    SS_USED         = 1,
    SS_UNLOCKED     = 3,
    SS_LOCKED       = 7
}TOReKSemaState;
```

Figura 3.13: Definição dos tipos enumerados `EOReKTaskType`, `EOReKTaskState` e `EOReKSemaState` do executivo *OReK*.

3.4.3.2 Definições Privadas ou Internas ao Executivo

No ficheiro de definições privadas do executivo *OReK* (`OReKPriv.h`) constam essencialmente descrições privadas internas ao executivo, e às estruturas anteriormente descritas `SOReKTaskCtrlBlk` e `SOReKSemaCtrlBlk`. É definido o tamanho mínimo absoluto da pilha das tarefas, tamanho da pilha auxiliar do executivo e valores limite de prioridade para os diversos tipos de tarefas suportadas pelo executivo OReK. A figura 3.14 ilustra as definições privadas do executivo *OReK*.

```
// Definitions
#define OREK_TASK_MIN_STACK_SZ              256u
#define OREK_KERN_AUX_STACK_SZ              256u

#define OREK_NON_RT_PRIO_LO_BOUND           OREK_NRT_PRIO_NEAR_RT
#define OREK_NON_RT_PRIO_UP_BOUND           OREK_NRT_PRIO_IDLE

#define OREK_NON_RT_BASE_PRIORITY           0xFFFFFFF8
#define OREK_SOFT_RT_BASE_PRIORITY          0x80000000
```

Figura 3.14: Definições privadas do executivo *OReK*.

3.4.4 Funções de Reacção às Interrupções do Temporizador e Periféricos

Nas versões *OReK-ARPA-Sw* ,*OReK-MB* e *OReK-PPC* são disponibilizadas em linguagem *C* as seguintes duas funções que respondem, do lado do executivo, às interrupções do temporizador e dos periféricos, sendo executadas no contexto das respectivas rotinas de serviço:

- `uint TimerCallback(uchar contxtNum, uint currentSP)` - reage às interrupções do temporizador, guardando o contexto da tarefa actual, actualizando o estado das tarefas, executando o algoritmo de escalonamento e lançando a próxima tarefa mais prioritária em execução;
- `int IntCallback(uchar contxtNum, uchar intSource)` - activa a tarefa aperiódica correspondente à interrupção gerada onde será feito o atendimento propriamente dito.

3.4.5 Escalonamento e Sincronização das Tarefas

A gestão de tarefas é baseada num conjunto de listas internas ao núcleo, nomeadamente:

- *TCBs* livres (`m_freeTCBs`);
- Tarefas paradas (`m_stoppedTaskList`);
- Tarefas periódicas inactivas (`m_idlePerTaskList`);
- Tarefas aperiódicas inactivas (`m_idleApeTaskList`);
- Tarefas ordinárias prontas a executar (`m_readyNonRTTaskList`);
- Tarefas de tempo-real não críticas prontas a executar (`m_readySoftRTTaskList`);
- Tarefas de tempo-real críticas prontas a executar (`m_readyHardRTTaskList`);
- Tarefas interrompidas (`m_preemptedTaskList`).

As listas `m_idlePerTaskList`, `m_readyNonRTTaskList`, `m_readySoftRTTaskList`, `m_readyHardRTTaskList` e `m_preemptedTaskList` possuem uma instância por cada contexto de execução do processador (variável no processador *ARPA-MT* e um nos outros casos). As restantes listas possuem apenas uma instância partilhada por todos os contextos de execução.

As listas `m_idlePerTaskList` e `m_idleApeTaskList` estão ordenadas pelo campo `m_activCnt` de forma a optimizar a activação das tarefas.

As listas de tarefas prontas a executar (`m_readyNonRTTaskList`, `m_readySoftRTTaskList` e `m_readyHardRTTaskList`) estão ordenadas pelo campo `m_priority` de forma a simplificar o escalonamento das tarefas.

A lista `m_preemptedTaskList` funciona como uma pilha de tarefas interrompidas de forma a que as tarefas sejam retomadas por ordem inversa à sua interrupção, simplificando desta forma a implementação do protocolo *SRP*.

As listas `m_freeTCBs` e `m_stoppedTaskList` não possuem qualquer ordenação sendo os *TCBs* colocados em geral no final (cauda) da lista e retirados do início (cabeça) da lista no primeiro caso e de forma arbitrária no segundo caso.

O contador `m_activCnt` dos *TCBs* pertencentes às listas ordenadas (com excepção da `m_readyNonRTTaskList`) é decrementado em todas as unidades de tempo. Quando atingir o valor zero, significa que, no caso das tarefas periódicas foi atingido o instante de uma nova instância e no caso das tarefas aperiódicas que foi cumprido o tempo mínimo entre activações consecutivas, pelo que a partir desse instante a tarefa pode, se necessário, ser activada.

O contador `m_priority` dos *TCBs* pertencentes à lista `m_readyHardRTTaskList` e dos *TCBs* relativos às tarefas de tempo-real críticas na lista `m_preemptedTaskList` é decrementado em todas as unidades de tempo. Quando atingir o valor 0, significa que foi alcançado o tempo limite de execução e se a tarefa ainda não tiver terminado, ocorreu uma violação temporal (*deadline missed*).

O escalonamento baseado em listas ordenadas é bastante simples, uma vez que se baseia unicamente no decremento de contadores e na inserção/remoção de elementos das listas. A preempção de uma tarefa dá-se quando forem cumpridos os requisitos definidos no capítulo 2 na discussão do protocolo *SRP*.

No início da operação do executivo *OReK* todas as listas estão vazias excepto a de *TCBs* livres. Esta situação é ilustrada na figura 3.15.

Durante a operação do sistema, as tarefas vão mudando de estado, sendo as listas alteradas em conformidade. Na figura 3.16 é ilustrada uma possível situação das listas correspondente aos seguintes estados das tarefas (assumindo um sistema com o máximo de 8 tarefas e um processador com um único contexto de execução):

- Tarefa 0 - *Preempted*;
- Tarefa 1 - *Ready*;
- Tarefa 2 - *Idle*;
- Tarefa 3 - *Running*;
- Tarefa 4 - *Stopped*;
- Tarefa 5 - *Idle*;
- Tarefa 6 - *Free*;
- Tarefa 7 - *Free*.

De notar que os *TCBs* representados nas diversas linhas das figuras 3.15 e 3.16 são os mesmos, isto é, tal como já foi referido acima, as listas são implementadas sobre uma única tabela estática de *TCBs*. Por uma questão de simplicidade apenas são mostradas nas figuras 3.15 e 3.16 uma lista *idle* e uma lista *ready*, embora na realidade existam várias consoante a criticalidade e o modo de activação das tarefas.

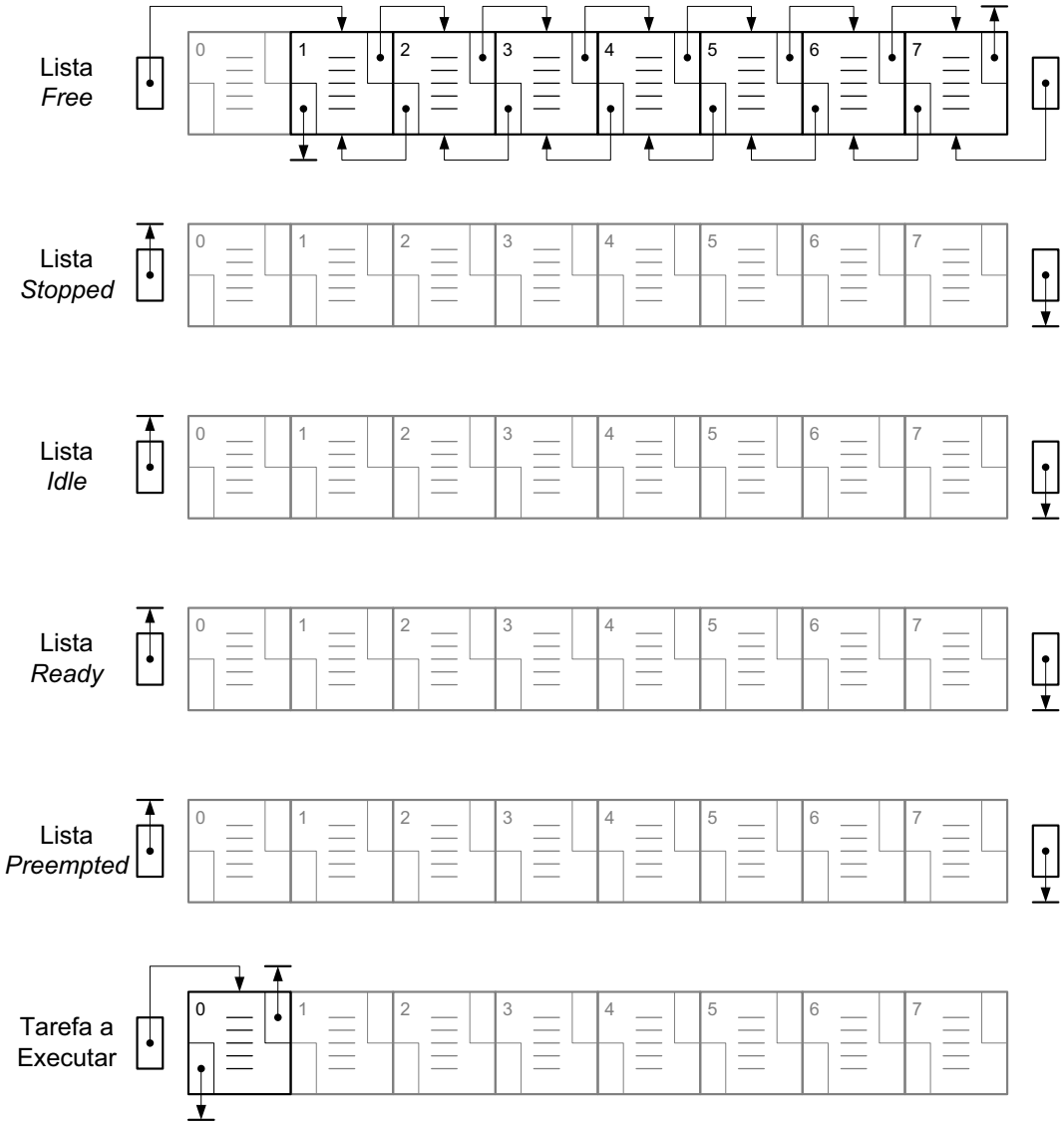


Figura 3.15: Estado inicial das listas de *TCBs* (retirado de [Arn07]).

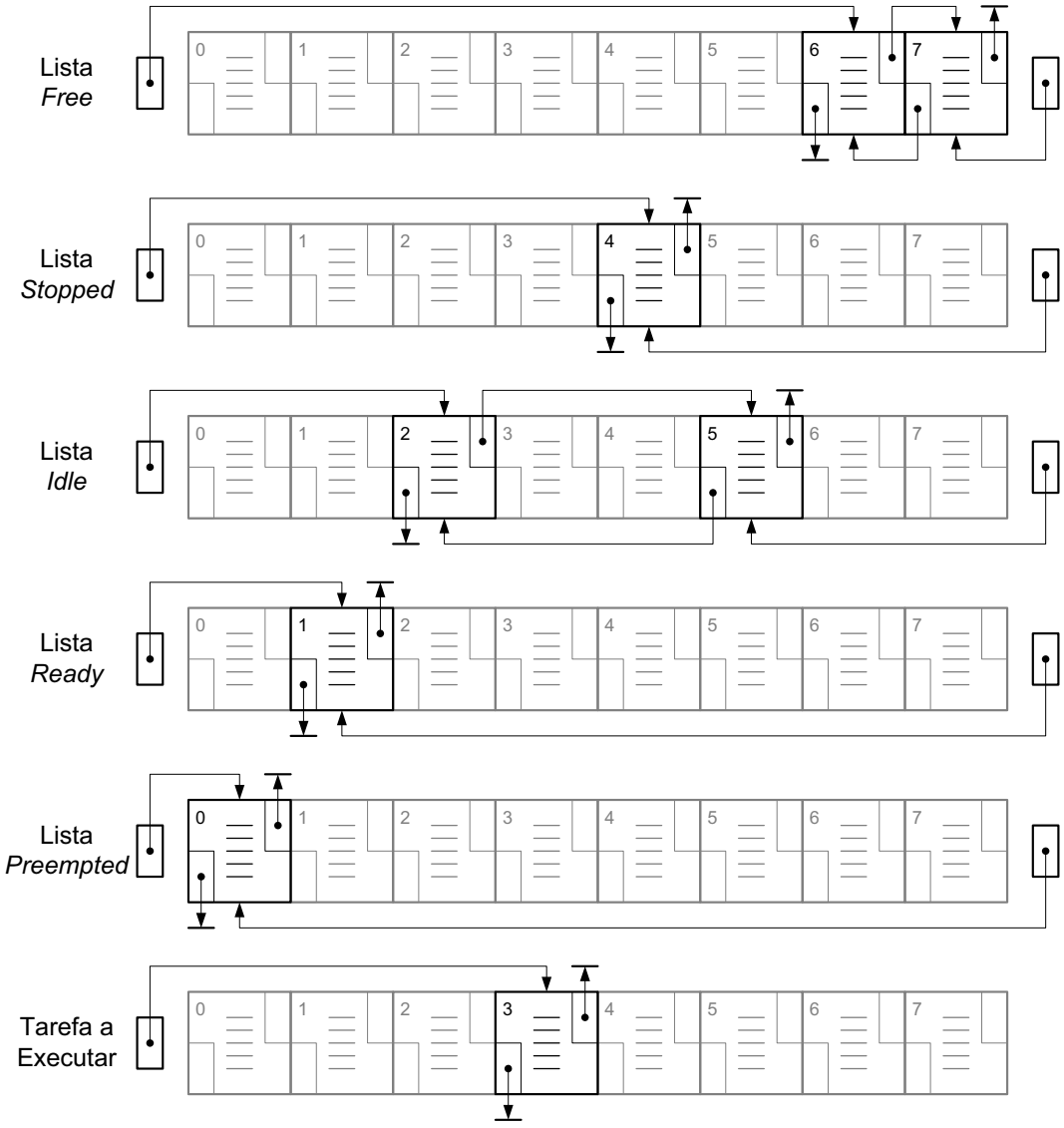


Figura 3.16: Possível estado das listas de *TCBs* em pleno funcionamento (retirado de [Arn07]).

3.4.6 A Estrutura de Ficheiros

Na figura 3.4 é mostrada a relação entre as diversas classes e estruturas de dados que constituem o executivo *OReK* e os ficheiros onde estão implementadas. De uma forma resumida, os ficheiros possuem o seguinte conteúdo:

Ficheiros públicos ou partilhados com a aplicação;

- `OReK.h` - ficheiro geral de interface para inclusão nas aplicações (C++/C);
- `OReKShrd.h` - contém definições partilhadas pelo executivo e pela aplicação (C++);
- `OReKKern.h` - contém a definição da classe `COReKKern` (C++);
- `OReKTask.h` - contém a definição da classe `COReKTask` (C++);
- `OReKTask.cpp` - contém a implementação da classe `COReKTask` (C++);
- `OReKTLst.h` - contém a definição e implementação da classe `COReKTaskList` (C++);
- `OReKSema.h` - contém a definição da classe `COReKSema` (C++);
- `OReKSema.cpp` - contém a implementação da classe `COReKSema` (C++);
- `OReKSLst.h` - contém a definição e implementação da classe `COReKSemaList` (C++);
- `MB-PPC.h` - contém a definição da interface do módulo específico das plataformas *MB-SoC* e *PPC-SoC* (C).

Ficheiros privados ou internos ao executivo;

- `OReKPriv.h` - contém definições internas (privadas) do executivo (C++);
- `OReKStrs.h` - contém a definição das mensagens intrínsecas do executivo (C);
- `OReKKern.cpp` - contém a implementação da classe `COReKKern` (C++);
- `OReKArch.h` - contém a definição da interface do módulo específico da plataforma (C);
- `OReKARPA.s` - contém a implementação do módulo específico da plataforma *ARPA-MT* (*assembly*);
- `OReKPC.asm` - contém a implementação do módulo específico da plataforma *Intel x86/MS-DOS* (*assembly*);
- `OReKXPS.cpp` - contém em linguagem de alto-nível parte da implementação do módulo específico da plataforma *Xilinx* para os processadores *MicroBlaze* e *PowerPC 405* (C);
- `OReKMBLo.s` - contém em linguagem de baixo-nível parte da implementação do módulo específico para o processador *MicroBlaze* (*assembly*);
- `xvectors.S` - contém em linguagem de baixo-nível parte da implementação do módulo específico para o processador *PowerPC 405* (*assembly*).

3.4.7 Versão Suportada pelo Coprocessador Cop2-OSC

É importante referir que esta subsecção é apresentada apenas para efeitos de comparação com as diversas versões implementadas, não tendo sido utilizada no contexto deste trabalho.

Tal como já foi dito, todas as implementações do executivo *OReK* possuem o mesmo modelo de programação e interface, simplificando desta forma o desenvolvimento e a portabilidade das aplicações.

O diagrama de blocos da figura 3.4, relativo à implementação integral em *software* é simplificado no caso da versão *OReK-ARPA-C2* (ver figura 3.17), sendo eliminadas as listas de tarefas e de semáforos e as funções invocadas no contexto das interrupções do temporizador e dos periféricos.

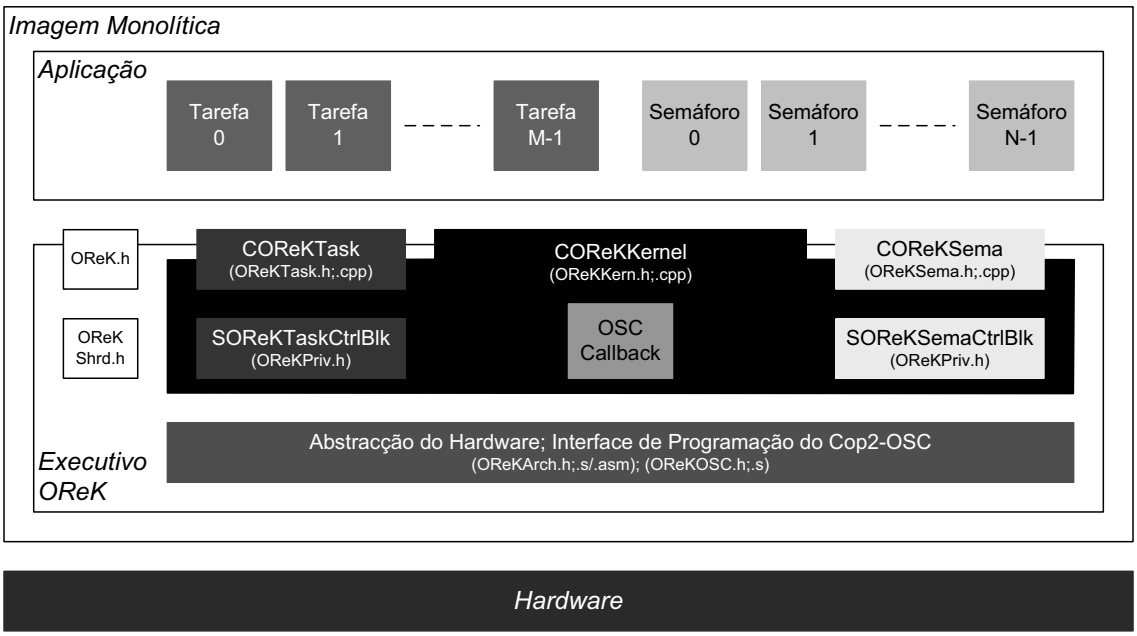


Figura 3.17: Estrutura da implementação suportada pelo coprocessador *Cop2-OSC* do executivo *OReK* (retirado de [Arn07]).

As estruturas de dados são também simplificadas uma vez que parte da informação que elas contêm nas implementações *OReK-ARPA-Sw*, *OReK-MB* e *OReK-PPC*, passa neste caso a residir no coprocessador *Cop2-OSC*. Mais concretamente, os campos delimitados pelas directivas `#ifndef __ARPA_OSC__` e `#endif // __ARPA_OSC__` do pré-processador indicadas nas figuras 3.6, 3.9 e 3.10 não são incluídos nesta versão.

Uma vez que o escalonamento das tarefas, a verificação das condições de preempção e a activação das tarefas aperiódicas para serviço de interrupções são completamente realizadas em *hardware* no coprocessador *Cop2-OSC*, nesta implementação existe apenas uma função invocada em caso de excepção para comutação da tarefa em execução. A função usada para este efeito possui o seguinte protótipo e é invocada no contexto da rotina de tratamento das excepções do coprocessador *Cop2-OSC*:

```
uint OSCCallback(uchar contxtNum, uint currentSP);
```

O módulo `OReKOSC` representado na figura 3.17, o qual contém a interface de programação do *Cop2-OSC* está escrito em *assembly* para que possa tirar partido das instruções especializadas e aceder aos registos disponíveis neste coprocessador.

3.5 Utilização em Aplicações

Finalmente e para concluir a descrição do executivo *OReK*, falta apenas descrever a sua utilização. O executivo *OReK* é disponibilizado na forma de ficheiros objecto ou bibliotecas, o que significa que deve ser associado com a aplicação durante a geração do módulo executável.

3.5.1 Ficheiros a Incluir

Tal como já foi referido acima, a construção de um sistema consiste na criação de um módulo executável monolítico que integra, quer a implementação das tarefas, quer o executivo de tempo-real. Do ponto de vista do utilizador, importa considerar os seguintes ficheiros:

Ficheiros de interface;

- `OReK.h` - ficheiro geral de interface que deve ser incluído nos módulos com código fonte da aplicação onde são invocadas funções do executivo;
- `MB-PPC.h` - ficheiro geral de interface que deve ser incluído nos módulos com código fonte da aplicação onde são invocadas funções das plataformas *MB-SoC* e *PPC-SoC*.

Uma das seguintes bibliotecas, consoante a implementação desejada;

- `OReK-PC.lib` - ficheiro de implementação da versão *Intel x86/MSDOS* que deve ser associado com a aplicação no caso da plataforma de implementação *Intel x86*;
- `OReK-ARPA-Sw.a` - ficheiro de implementação da versão *ARPA-MT* sem suporte do coprocessador *Cop2-OSC* e que deve ser associado com a aplicação no caso da plataforma de implementação *ARPA-Sw*;
- `OReK-ARPA-C2.a` - ficheiro de implementação da versão *ARPA-MT* com suporte do coprocessador *Cop2-OSC* e que deve ser associado com a aplicação no caso da plataforma de implementação *ARPA-C2*;
- `OReK-MB.a` - ficheiro de implementação da versão *MB-SoC* que deve ser associado com a aplicação no caso da plataforma de implementação *MB-SoC*;
- `OReK-PPC.a` - ficheiro de implementação da versão *PPC-SoC* que deve ser associado com a aplicação no caso da plataforma de implementação *MB-SoC*.

3.5.2 Requisitos de Memória

Os requisitos de memória do executivo *OReK* são aproximadamente os seguintes:

- Versão *OReK-ARPA-Sw* - $20K + (48 \times \#Tarefas) + (16 \times \#Semáforos)$ bytes;
- Versão *OReK-ARPA-C2* - $10K + (16 \times \#Tarefas) + (8 \times \#Semáforos)$ bytes;
- Versão *OReK-MB* - $50K + (92 \times \#Tarefas) + (32 \times \#Semáforos)$ bytes;
- Versão *OReK-PPC* - $80K + (64 \times \#Tarefas) + (32 \times \#Semáforos)$ bytes.

Na plataforma *ARPA* a compilação do executivo *OReK* e das aplicações é suportado por um conjunto de ficheiros de projecto (*Makefiles*) desenvolvidas com o objectivo de simplificar este procedimento.

Nas plataformas *MB-SoC* e *PPC-SoC* a compilação do executivo *OReK* é gerida automaticamente por ficheiros (*Makefiles*) gerados pela ferramenta *EDK* (*Embedded Development Kit*) da *Xilinx* [Xil08h].

3.5.3 Exemplo

A figura 3.18 ilustra um exemplo de um sistema de tempo-real implementado com o executivo *OReK*. A sua prototipagem foi realizada numa *FPGA XC2VP30-7ff896* da família *Virtex-II Pro* da *Xilinx* [Dig08].

O sistema possui cinco tarefas idênticas (diferentes instâncias da classe `CDemoTask`). Cada tarefa inverte e envia dados via porta série, com as características temporais especificadas pelo utilizador.

A definição da tarefa `CDemoTask` encontra-se na primeira metade da figura 3.18 e possui dois atributos, um construtor e o método de entrada. O primeiro atributo é uma máscara usada na inversão do bit sobre o qual a tarefa opera. O segundo atributo é uma cópia do valor actual da saída de dados da porta série. No construtor, com base no parâmetro fornecido pelo utilizador e que identifica o bit sobre o qual a tarefa actua, é calculada a máscara referida acima. No método de entrada é lido o valor actual de todos os bits de dados da porta, invertido o bit pretendido e enviado o novo valor via porta série.

A região crítica da tarefa está protegida por um semáforo binário que garante o seu acesso em regime de exclusão mutua. Desta forma cada tarefa apenas envia invertido o bit que lhe corresponde, e por conseguinte, é assegurado o correcto funcionamento do sistema de tempo-real.

O programa principal (função `main`) começa por instanciar cinco objectos (`task1` a `task5`). Seguidamente inicializa o executivo *OReK* (função `COReKKern::Initialize(...)`). Os argumentos desta função correspondem aos seguintes parâmetros de configuração:

- Número máximo de tarefas - 10;
- Número máximo de semáforos - 2;
- Frequência do sistema - 100000 KHz;

```
// Includes
#include <stdio.h>
#include "OReK.h"
#if defined (__MICROBLAZE__) || defined (__PPC__)

    #include "MB-PPC.h"

#endif // __MICROBLAZE__ || __PPC__

// Defines
#define SYS_FREQ            100000  //(Khz)
#define TICK_RESOLUTION     100     //(us)
#define MAX_TASKS           10
#define MAX_SEMAS           2

#define RS232_UART_BASEADDR 0x40600000


class CDemoTask : public COReKTask
{
public:
    CDemoTask(uchar bit)
    {
        m_xorValue = 1 << bit;
    }


public:
    virtual void Main()
    {
        // Lock sema
        DemoTaskSema.Wait();

        m_portValue ^= m_xorValue;

        XUartLite_SendByte(RS232_UART_BASEADDR, m_portValue);

        // Unlock sema
        DemoTaskSema.Signal();
    }

protected:
    uchar m_xorValue;
    static uchar m_portValue;
};

uchar CDemoTask::m_portValue = 0x00;

COReKSema DemoTaskSema;


// main
int main(int argc, char* argv[])
{

    CDemoTask task1(0);
    CDemoTask task2(1);
    CDemoTask task3(2);
    CDemoTask task4(3);
    CDemoTask task5(4);
```

```
    print("\r\nInitializing OReK ...");
    if (COReKKern::Initialize(MAX_TASKS, MAX_SEMAS,
         SYS_FREQ, TICK_RESOLUTION) != OREK_INFO_SUCCESS)
    {
        print("Failed\r\n");
        return 1;
    }
    print("OK\r\n");

    print("Creating tasks ...\r\n");
    if ((task1.CreateSoftRTPer(2, 2) != OREK_INFO_SUCCESS)    ||
        (task2.CreateSoftRTApe(16, 2) != OREK_INFO_SUCCESS)   ||
        (task3.CreateHardRTPer(4, 4, 4) != OREK_INFO_SUCCESS) ||
        (task4.CreateHardRTApe(1, 1, 1) != OREK_INFO_SUCCESS) ||
        (task5.CreateNonRT(7, true) != OREK_INFO_SUCCESS))
    {
        print("Failed\r\n");
        COReKKern::ShutDown();
        return 2;
}
    print("OK\r\n");

    print("Creating Semaphores\r\n");
    DemoTaskSema.Create();

    print("Registering Tasks\r\n");
    DemoTaskSema.RegisterTask(task1);
    DemoTaskSema.RegisterTask(task2);
    DemoTaskSema.RegisterTask(task3);
    DemoTaskSema.RegisterTask(task4);
    DemoTaskSema.RegisterTask(task5);

    print("Enabling Semaphores\r\n");
    if(DemoTaskSema.Enable() != OREK_INFO_SUCCESS)
    {
        print("Failed\r\n");
        COReKKern::ShutDown();
        return 3;
    }
    print("OK\r\n");
    print("StartAllTasks...\r\n");
    if(COReKKern::StartAllTasks() != OREK_INFO_SUCCESS)
    {
        print("Failed\r\n");
        COReKKern::ShutDown();
        return 4;
    }
    print("OK\r\n");

    while (COReKKern::GetTickCount() < 10000u)
    {
        // Idle time
    }

    print("Exiting\r\n");
    return COReKKern::ShutDown();
}
```

Figura 3.18: Exemplo de um sistema implementado sobre o executivo *OReK*.

- Resolução temporal - 100 micro-segundos.

Em caso de erro na inicialização o programa termina, caso contrário são criadas as tarefas. A existência de funções para criar as respectivas tarefas simplifica a escrita do construtor da tarefa, uma vez que os parâmetros temporais das tarefas são passados directamente ao núcleo. Além disso, facilita também a sinalização de situações de erro.

No exemplo em concreto são ilustrados os diversos tipos de tarefas suportadas pelo executivo *OReK*, estas são de tempo-real não críticas (periódicas ou aperiódicas), tempo-real críticas (periódicas ou aperiódicas) e ordinárias.

No caso das tarefas periódicas de tempo-real não críticas, são especificados os parâmetros período e fase inicial. Para as tarefas aperiódicas de tempo-real não críticas é explicitado o intervalo mínimo entre activações consecutivas (*MIT*) e o número da respectiva entrada do sinal interrupção.

Nas tarefas periódicas de tempo-real críticas são detalhados os parâmetros período, *deadline* relativa e fase inicial. Relativamente às tarefas aperiódicas de tempo-real críticas é definido o *MIT*, a *deadline* relativa e ainda a fase inicial.

Por último, para as tarefas ordinárias de baixa prioridade, os parâmetros necessários a especificar são a sua prioridade base e se estas executam uma única vez e terminam (*single shot*) ou ficam a executar durante o tempo livre do processador.

Seguidamente é criado o semáforo e são registadas todas as tarefas que acedem ao mesmo recurso partilhado do sistema. Este registo é necessário devido à utilização da política *SRP*. Esta necessita de ter conhecimento das características temporais das tarefas para calcular o respectivo tecto.

De seguida o semáforo é activado e em caso de sucesso segue-se o arranque de todas as tarefas. A partir deste momento as tarefas são executadas concorrentemente com o programa principal.

Finalmente, quando tiverem sido completados 10000 ciclos de execução *(ticks)*, o programa principal chama a função de terminação do executivo, terminando também de seguida.

Capítulo 4

Avaliação do Desempenho

Sumário

Este capítulo apresenta a avaliação do desempenho do executivo de tempo-real *OReK* nas diversas plataformas em que é suportado (quer as resultantes deste trabalho, *MB-SoC* e *PPC-SoC*, quer as apresentadas em [Arn07] para efeitos de comparação).

O capítulo começa com a descrição dos aspectos que se pretende avaliar. De seguida introduz os pré-requisitos necessários para efectuar a avaliação e apresenta os tempos de processamento de uma aplicação baseada no executivo *OReK*. Seguidamente são apresentados os tempos de execução das seguintes funções internas do executivo *OReK*: salvaguarda e restauro do contexto de uma tarefa; processamento periódico e comutação de tarefas; selecção da próxima tarefa a executar; início de uma instância de uma tarefa; terminação de uma instância de uma tarefa; activação de uma tarefa aperiódica por uma interrupção.

São ainda indicados os tempos de execução dos seguintes serviços disponibilizados à aplicação pelo executivo *OReK*: inicialização do executivo; leitura do temporizador do sistema; activação e desactivação da preempção das tarefas; criação e destruição de uma tarefa; arranque e paragem de uma tarefa; activação explícita de uma tarefa aperiódica; leitura do estado de uma tarefa; criação e destruição de um semáforo; registo de uma tarefa num semáforo; activação de um semáforo; bloqueio e libertação de um semáforo. O capítulo conclui com a análise dos resultados obtidos e com um resumo dos aspectos não avaliados.

4.1 Introdução

Uma avaliação adequada das arquitecturas e implementações, quer ao nível do *hardware*, quer ao nível do *software* de sistemas computacionais especializados para aplicações embutidas de tempo-real construídas com base em sistemas integrados reconfiguráveis, deverá cobrir pelo menos os seguintes aspectos:

- Estudo do impacto, ao nível dos recursos de implementação, tempo de execução e consumo de potência, da utilização de um processador embutido, na execução de uma aplicação composta por diversas tarefas concorrentes;
- Comparação entre um processador implementado em lógica dedicada e interno à *FPGA* com processadores implementados em blocos lógicos configuráveis;
- Avaliação dos requisitos computacionais e da eficiência energética associados ao funcionamento do executivo de tempo-real *OReK*, quando a sua implementação é realizada em diversos sistemas integrados com unidades de processamento distintas;
- Comparação do ponto anterior com valores de referência, associados à execução de um núcleo ou executivo de tempo-real, quando a sua implementação é realizada completamente em *software* e ainda, quando parte das funções são suportadas por *hardware* dedicado.

No entanto, uma avaliação completa das implementações realizadas no âmbito deste trabalho constitui por si só uma tarefa bastante complexa e morosa. Isto deve-se ao carácter especializado dos sistemas embutidos, pelo que certos parâmetros e características dos sistemas concebidos neste trabalho só fazem sentido ser avaliadas para aplicações em concreto.

Como o tempo para o fazer é limitado, a avaliação apresentada neste capítulo será restringida a parte dos aspectos referidos acima, sendo dada especial importância aos terceiro e quarto pontos apresentados.

Mais concretamente, alguns dos aspectos mais importantes a avaliar são os seguintes:

- Medição do tempo de execução dos serviços e primitivas disponibilizadas à aplicação por um executivo de tempo-real, quando a sua implementação é realizada em dois sistemas integrados com processadores com diferentes capacidades de processamento;
- Comparação das taxas de ocupação do processador devido às funções internas do executivo de tempo-real *OReK* sobre o qual a aplicação está implementada, nas duas situações referidas no ponto anterior. Neste ponto interessa avaliar os tempos de comutação e escalonamento das tarefas, assim como os requisitos computacionais das operações de manutenção internas do executivo, de forma a determinar a sua carga computacional relativa em função do período do sistema;
- Análise do comportamento do sistema quando o serviço de interrupções é efectuado através de tarefas aperiódicas activadas por *software* utilizando os mecanismos tradicionais implementados nos sistemas integrados desenvolvidos. Neste ponto interessa avaliar a latência no atendimento das interrupções e as oscilações temporais (*jitter*) na execução das tarefas aperiódicas que as servem e nas restantes actividades periódicas do sistema.

Do ponto de vista do projectista de um sistema de tempo-real, estes são alguns dos aspectos mais importantes que condicionam a escolha da plataforma de implementação.

4.2 Pré-requisitos

Para que a avaliação do desempenho, segundo os aspectos expostos na secção anterior, possa ser feita de uma forma simples, sistemática e rigorosa é fundamental dispor de plataformas com as capacidades adequadas. Estas serão baseadas nos processadores *MicroBlaze* e *PowerPC*, sobre as quais pode ser implementada uma aplicação embutida.

Tais plataformas devem disponibilizar os recursos de *hardware* e de *software* adequados à aplicação alvo e os mecanismos que garantam uma boa observabilidade e uma configuração flexível de forma a simplificar a avaliação que se pretende efectuar.

4.2.1 Requisitos de Hardware

Do ponto de vista do *hardware*, as plataformas devem possuir capacidades de interface com o mundo exterior de forma a permitir a implementação de um sistema embutido realista. Para este efeito foram desenvolvidos e implementados os sistemas integrados *MB-SoC* e *PPC-SoC*. Estes possuem em comum os seguintes componentes:

- Memória interna;
- Interface para memória externa;
- Unidade de temporizadores/contadores;
- Controlador de interrupções;
- *UART RS232*;
- Portos de entrada/saída;
- Módulo de depuração;
- Módulo de temporizadores especificamente construídos para medição precisa de intervalos de tempo.

O sistema integrado *MB-SoC* possui ainda o processador *MicroBlaze*, enquanto que o *PPC-SoC* possui o processador *PowerPC 405*. As prototipagem foram realizadas numa *FPGA XC2VP30-7ff896* da família *Virtex-II Pro* da *Xilinx* [Dig08]. A figura 4.1 apresenta a placa de desenvolvimento utilizada e explicita os diversos componentes constituintes. Ambos os sistemas integrados são descritos em maior detalhe no apêndice B.

4.2.2 Requisitos de Software

Os serviços devem ser prestados à aplicação de forma homogénea, tornando transparentes as especificidades do *hardware* de suporte. Isto é, todos os serviços devem ser prestados independentemente da execução ser efectuada na plataforma *MB-SoC* ou *PPC-SoC*.

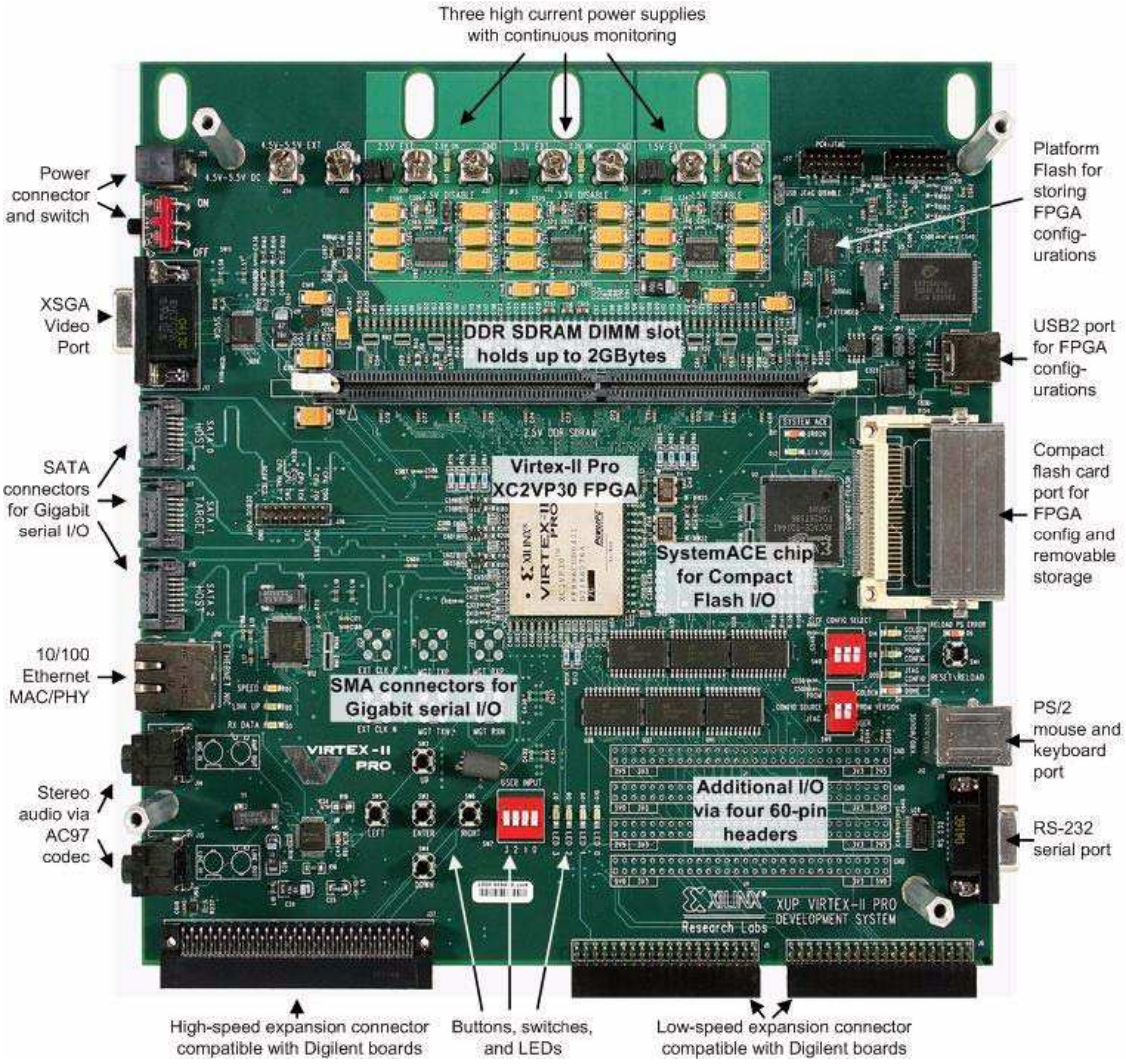


Figura 4.1: Fotografia da placa de desenvolvimento utilizada (retirado de [Dig08]).

Do ponto de vista da avaliação do desempenho é fundamental a utilização de temporizadores que possam medir intervalos de tempo de forma precisa e com carga computacional idealmente nula. Assim, foi desenvolvido um módulo com diversos temporizadores ou contadores que permitem, de uma forma simples e com sobrecarga computacional bastante reduzida, contar o número de ciclos de relógio gastos em qualquer função do sistema ou da aplicação. O módulo desenvolvido apresenta a mesma interface de *software* para ambas as implementações (*MB-SoC* e *PPC-SoC*), sendo apresentado em maior detalhe no apêndice C.

4.3 Tempos de Processamento de Uma Aplicação Baseada no OReK

O tempo de processamento em cada período (T_{Tick}) ou unidade temporal (*tick*) num sistema baseado no núcleo *OReK* pode ser dividido em três componentes:

- $T_{OReK}(i)$ - tempo de execução do *OReK*;
- $T_{Appl}(i)$ - tempo de execução da aplicação;
- $T_{Idle}(i)$ - tempo livre ou de inactividade.

Estas estão relacionadas pela seguinte expressão:

$$T_{Tick} = T_{OReK}(i) + T_{Appl}(i) + T_{Idle}(i)$$

Em que i representa a i-ésima unidade temporal, uma vez que estas componentes variam em cada unidade temporal.

O processamento realizado pelo núcleo *OReK* deve-se essencialmente aos seguintes dois aspectos:

- Funções internas do executivo para actualização das suas estruturas de dados internas, gestão temporal, escalonamento, sincronização, lançamento, comutação e terminação das tarefas;
- Serviços iniciados pela aplicação e realizadas no contexto da mesma para configuração e obtenção do estado do núcleo e manipulação das tarefas e dos semáforos.

Assim, o tempo de execução do *OReK* ($T_{OReK}(i)$) pode ser subdividido em três componentes: $T_{Fint}(i)$, $T_{Serv}(i)$ e T_{Swit}, relacionadas da seguinte forma:

$$T_{OReK}(i) = T_{Fint}(i) + T_{Serv}(i) + T_{Swit}$$

A componente $T_{Fint}(i)$ é devida às funções internas do núcleo variando com o número de tarefas e de semáforos bem como com o seu estado em particular num dado instante. Além disso, depende também de factores específicos da aplicação, tais como o ritmo a que são iniciadas e terminadas as tarefas ou a taxa de activação das interrupções dos periféricos.

A componente $T_{Serv}(i)$ deve-se aos serviços iniciados pela aplicação. Alguns dos serviços possuem tempos de execução fixos, enquanto outros dependem do número e do estado das tarefas e dos semáforos do sistema.

Finalmente, a componente T_{Swit} deve-se à salvaguarda e restauro dos registos ou contexto de uma tarefa, sendo constante e independente do estado interno do executivo e função apenas da dimensão do contexto do processador necessário salvaguardar para cada tarefa.

Embora com diferentes pesos, estas três componentes aplicam-se, quer às implementações *OReK-MB*, *OReK-PPC* e *OReK-ARPA-Sw*, quer à *OReK-ARPA-C2*.

4.4 Avaliação do Tempo de Execução

A avaliação temporal do executivo *OReK* visa obter os majorantes dos tempos acima descritos, $T_{Fint}(i)$, $T_{Serv}(i)$ e T_{Swit}, de forma avaliar o seu impacto na carga computacional do sistema. A avaliação será feita com os sistemas integrados *MB-SoC* e *PPC-SoC*.

4.4.1 Parâmetros do Processador MicroBlaze

O processador *MicroBlaze* possui os seguintes parâmetros de configuração:

- *HW_VER = 6.00.b* (versão do núcleo de propriedade intelectual);
- *C_DEBUG_ENABLED = 1* (activação da depuração);
- *C_NUMBER_OF_PC_BRK = 2* (número de pontos de paragem do módulo de depuração);
- *C_NUMBER_OF_RD_ADDR_BRK = 1* (número de pontos de paragem do módulo de depuração relativos a endereços onde são efectuadas leituras);
- *C_NUMBER_OF_WR_ADDR_BRK = 1* (número de pontos de paragem do módulo de depuração relativos a endereços onde são efectuadas escritas);
- *C_USE_BARREL = 1* (utilização do módulo *barrel shifter*);
- *C_USE_DIV = 1* (habilita a realização de divisões em *hardware*;
- *C_FAMILY = virtex2p* (família da *FPGA*);
- *C_FSL_LINKS = 2* (número interfaces para ligações *FSL*).

Ainda relativamente ao processador *MicroBlaze*, foi especificada uma frequência de funcionamento de *110 MHz*.

4.4.2 Parâmetros do Processador PowerPC 405

O processador *PowerPC 405* foi configurado com os seguintes parâmetros:

- *HW_VER = 2.00.c* (versão do núcleo de propriedade intelectual);
- *C_DISABLE_OPERAND_FORWARDING = 1* (desactiva o encaminhamento do operando para as instruções de leitura da memória);
- *C_DETERMINISTIC_MULT = 0* (multiplicador implementado em blocos de lógica dedicada);
- *C_MMU_ENABLE = 1* (activação da unidade de gestão de memória);
- *C_DCR_RESYNC = 0* (ressincronização do barramento *Device Control Register*);

Em relação ao processador *PowerPC 405*, foi ainda especificada uma frequência de operação de *300 MHz*.

4.4.3 Parâmetros do Processador ARPA

Os valores de referência associados à execução do núcleo ou executivo de tempo-real *OReK* no sistema integrado *ARPA-SoC* foram retirados de [Arn07]. Na avaliação efectuada, o processador *ARPA-MT* foi sintetizado e implementado com os seguintes parâmetros:

- *NUM_CONTEXTS = 1* (número de contextos de execução);
- *CYCLIC_INST_ISSUE = false* ("true" para escalonamento round-robin das instruções e "false" para escalonamento baseado em prioridades);
- *DYN_CONTEXT_PRIO = false* ("true" para escalonamento das instruções com base em prioridades dinâmicas e "false" para prioridades fixas);
- *EX_STAGE_LATENCY = 1* (latência da etapa *EX*);
- *FAST_WB_STAGE = false* ("true/false" para etapas *WB* rápidas/lentas);
- *EX_ENABLE_FWD = true* (encaminhamento da etapa *EX* do *pipeline*);
- *MA_ENABLE_FWD = true* (encaminhamento da etapa *MA* do *pipeline*);
- *WB_ENABLE_FWD = true* (encaminhamento da etapa *WB* do *pipeline*);
- *ADV_REGISTER_CMP = false* (comparação especulativa dos registos em paralelo com a detecção de dependências e com o encaminhamento);
- *EMUL_INT_DIVISION = true* (inclui os circuitos que implementam a emulação suportada por hardware das divisões de números inteiros);
- $LOG2\_TASK\_TAB\_SIZE = log_2(\#Tarefas)$ (logaritmo na base 2 da dimensão da tabela de tarefas);
- *LOG2_MAX_PREPTD_TASKS = LOG2_TASK_TAB_SIZE* (logaritmo na base 2 do número máximo de tarefas interrompidas por contexto de execução);
- *LOG2_TASK_BLK_SIZE = 4* (logaritmo na base 2 da dimensão do bloco de tarefas usado para efeitos de escalonamento);
- *CONTEXT_SHARED_SCHED = false* (implementação de um escalonador de tarefas partilhado por todos os contextos de execução ou específico de cada contexto);
- *SEMA_UNIT_IMPLEMENT = true* (implementação ou supressão do módulo *Semaphore Handling Unit*);
- $LOG2\_SEMA\_TAB\_SIZE = log_2(\#Semáforos)$ (logaritmo na base 2 da dimensão da tabela de semáforos);
- *LOG2_MAX_LOCKED_SEMAS = LOG2_SEMA_TAB_SIZE* (logaritmo na base 2 do número máximo de semáforos bloqueados por contexto de execução).

O executivo *OReK* foi executado no processador *ARPA-MT* a operar a 24 *MHz*.

4.4.4 Abordagens Utilizadas na Avaliação do OReK

Os valores de referência obtidos do sistema integrado *ARPA-SoC* estão associados à execução do núcleo ou executivo de tempo-real *OReK*, implementado nas seguintes variantes:

1. O executivo *OReK* opera integralmente em *software*, sendo designado por *OReK-ARPA-Sw*;
2. Parte das funções internas do executivo *OReK* são suportadas por uma unidade de coprocessamento em *hardware* (*Cop2-OSC*) [Arn07] e as restantes funções do executivo são executadas em *software*, sendo neste caso designado por *OReK-ARPA-C2*.

A avaliação do executivo *OReK* a operar integralmente em *software* foi efectuada primeiramente na plataforma *MB-SoC*. Nesta foram empregues duas abordagens:

1. A memória interna da *FPGA* é utilizada para armazenar a informação do módulo monolítico resultante da compilação do executivo mais a aplicação, sendo esta implementação designada por *OReK-MB-Int*. É ainda importante referir que o acesso a esta memória é efectuado através de um barramento dedicado que interliga o processador e a memória;
2. A memória interna referida no ponto anterior é utilizada para armazenar apenas os segmentos de código (*text*) e dados estáticos não inicializados (*data*). Os segmentos de dados dinâmicos (*heap*) e pilha (*stack*) são armazenados em memória interna à *FPGA* mas ligada ao processador através do barramento *OPB* (*On-Chip Peripheral Bus*) [Xil08g], tendo esta implementação sido designada por *OReK-MB-OPB*. Esta segunda abordagem utiliza um barramento que não é dedicado para transferências de informação entre processador e memória, pelo que o tempo de acesso à memória será maior. Facto que contribui para um aumento dos parâmetros temporais do executivo *OReK*, resultando numa degradação temporal.

Na realização da avaliação temporal do executivo *OReK* integrado na plataforma *PPC-SoC*, foram utilizadas as seguintes três abordagens:

1. A memória interna da *FPGA* é utilizada para armazenar a informação do módulo monolítico resultante da compilação do executivo mais a aplicação, sendo esta implementação designada por *OReK-PPC-Int*. Esta é uma implementação análoga à do ponto 1 na plataforma *MB-SoC*, ou seja, o acesso a esta memória é efectuado através de um barramento dedicado que interliga o processador e a memória;
2. A memória interna referida no ponto anterior é utilizada para armazenar apenas os segmentos *text* e *data*. Os segmentos *heap* e *stack* são armazenados em memória *DDR* (*Double Data Rate*) externa, sem *cache*. Esta implementação foi designada por *OReK-PPC-DDR*;
3. Implementação análoga à do ponto anterior, porém utiliza memória (*cache*) do processador *PowerPC 405* para optimizar o acesso à memória externa, sendo designada por *OReK-PPC-Cached*. De referir que esta implementação contribui para uma optimização de desempenho à custa da introdução de indeterminismo no sistema.

A motivação para o uso destas abordagens está relacionada com o estudo do impacto da variação da carga computacional do executivo *OReK*, introduzido pelo acesso a dados em memórias com diferentes latências.

A razão pela qual os segmentos *text* e *data*, para as diversas abordagens implementadas, estarem armazenados em memória interna, deve-se grandemente a questões relacionadas com a simplicidade de implementação. De facto, para implementar as abordagens anteriores é suficiente recorrer à ferramenta *linker script* e especificar o local onde residem os respectivos segmentos. Por outro lado, para o executivo *OReK* operar com toda a informação do módulo monolítico em memória externa, seria no mínimo necessário utilizar um *bootloader* para inicialização da memória. Uma implementação possível passaria por adicionar um módulo de *interface* com uma memória do tipo não volátil em ambas as plataformas de *hardware* desenvolvidas [Xil08a]. Tal memória seria posteriormente utilizada para armazenar o módulo monolítico e no momento de arranque do sistema, um *bootloader* inicializaria a respectiva memória externa.

Por último, o *software* de ambos os sistemas integrados foi compilado no modo *Release*, definido na ferramenta *SDK* (*Xilinx Software Development Kit*), ou seja, sem qualquer nível de depuração e optimização elevada (*-O3*).

Apresentadas as diversas implementações, é então conveniente referir que a avaliação do tempo de processamento das funções internas e dos serviços disponibilizados à aplicação deve ser realizada nas condições mais desfavoráveis de forma a obter-se o respectivo majorante.

Ao nível do processamento periódico e escalonamento de tarefas, nos executivos em *software* o caso mais desfavorável ocorre geralmente quando, no início de uma nova unidade temporal, existe a activação simultânea de todas as tarefas do sistema, encontrando-se todas no estado *idle* e devendo todas transitar para o estado *ready*. Dito de outra forma, a lista *idle* encontra-se completamente cheia sendo esvaziada e passando a lista *ready* a estar completamente preenchida. Esta é também a situação que acontece no executivo *OReK*.

Os tempos apresentados nas próximas secções foram obtidos por um dos seguintes métodos, consoante a complexidade e/ou a ordem de grandeza do tempo a medir:

- Medição com um dos temporizadores referidos na secção 4.2.2;
- Análise directa do código *assembly*;
- Manipulação do código fonte do núcleo de forma colocar as tarefas e os semáforos nos estados pretendidos e invocação directa das funções que se pretende avaliar.

Por último, os resultados das tabelas apresentadas nas próximas secções poderiam estar normalizados para facilitar a compreensão. Por outro lado, a normalização iria mascarar as diferenças na frequência de operação máxima de cada um dos processadores e serviria apenas para comparar as eficiências dos conjuntos de instruções e dos compiladores.

4.4.5 Funções Internas do Executivo

As funções internas do executivo *OReK* realizadas periodicamente, despoletadas por eventos externos (interrupções) ou iniciadas de forma não explícita pela aplicação são as seguintes:

- Salvaguarda e restauro do contexto de uma tarefa;
- Processamento periódico e comutação de tarefas;
- Selecção da próxima tarefa a executar;
- Início de uma instância de uma tarefa;
- Terminação de uma instância de uma tarefa;
- Activação de uma tarefa aperiódica por uma interrupção.

Os tempos de execução de cada uma destas funções nas implementações anteriormente referidas serão apresentados nas próximas subsecções.

4.4.5.1 Salvaguarda e Restauro do Contexto de uma Tarefa

A tabela 4.1 mostra o tempo de salvaguarda e restauro do contexto de uma tarefa nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

O tempo de salvaguarda e restauro do contexto de uma tarefa é invariante do estado interno da *CPU*. Sendo estas operações realizadas completamente em *software*, o tempo que demoram depende apenas das instruções executadas e da frequência de operação do processador.

O tempo na implementação *OReK-ARPA-C2* é ligeiramente superior ao da *OReK-ARPA-Sw* devido aos registos adicionais do coprocessador que devem ser salvaguardados durante uma comutação de tarefas. No entanto, a diferença é marginal e largamente compensada pelo menor número de comutações de tarefas na implementação *OReK-ARPA-C2*. O tempo na implementação *OReK-MB-Int* é proporcional à relação entre as frequências de funcionamento dos processadores *MicroBlaze* e *ARPA*. O tempo na implementação *OReK-MB-OPB* é consideravelmente superior, o que se explica pelo facto do barramento *OPB* introduzir uma latência considerável no acesso à memória.

Relativamente ao tempo na implementação *OReK-PPC-Int*, este é maior do que seria de esperar mas pode ser justificado por um conjunto de agravantes:

- O número de registos que é necessário salvaguardar e restaurar é ligeiramente maior do que nas implementações anteriores (40 nesta implementação, 28 para o processador *ARPA* e 30 para o *MicroBlaze*);
- Ao contrário das outras implementações, a *CPU* e a memória têm frequências de operação diferentes. O facto do processador *PowerPC 405* estar implementado em lógica dedicada, possibilita frequências de operação que não são atingíveis por outros circuitos implementados em lógica programável. Nesta implementação, à excepção do processador, os módulos do sistema (incluindo a memória interna da *FPGA*) operam à frequência de *100 MHz*. Como a *CPU* opera a *300 MHz*, é imediato concluir que vai existir uma latência maior do que seria esperado na transferência de informação entre a *CPU* e a memória interna da *FPGA*.

O tempo na implementação *OReK-PPC-DDR* é bastante maior do que na implementação com memória interna (quase oito vezes superior). Este é devido a razões análogas à implementação *OReK-MB-OPB*, com a agravante da penalização temporal no acesso ao conteúdo da memória *DDR* ser ainda maior. Na verdade, para aceder à memória *DDR* é necessário passar pelo barramento *PLB* (*Processor Local Bus*) e ainda pelo controlador da memória *DDR*. Este envolve uma série de operações (descodificação de endereços de linha e de coluna, sinais de controlo, etc.) e somente depois é possível a transferência de informação entre a *CPU* e a memória *DDR*.

Implementação	**Tempo de Execução** (μs)
OReK-ARPA-Sw	3,2
OReK-ARPA-C2	3,4
OReK-MB-Int	0,7
OReK-MB-OPB	3,4
OReK-PPC-Int	1,4
OReK-PPC-DDR	11,0
OReK-PPC-Cached	0,8

Tabela 4.1: Tempos de salvaguarda e restauro do contexto de uma tarefa no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Na implementação *OReK-PPC-Cached* são claros os ganhos introduzidos pela utilização de uma memória *cache* intermédia. O tempo de salvaguarda e restauro do contexto passou a ser bastante inferior à implementação anterior (quase catorze vezes menor), sendo ainda inferior à implementação *OReK-PPC-Int*. A justificação para este tempo obtido está relacionada com o facto do processador *PowerPC 405* possuir uma unidade de *cache* interna. Na verdade possui duas unidades, uma para dados e outra para instruções. Estas operam à frequência do processador o que permite uma baixa latência na transferência de informação. De referir que um *cache hit* possibilita um desempenho idêntico ao acesso à memória interna da *FPGA*, sendo este efectuado através de um barramento dedicado que interliga apenas o processador e a memória [Xil08i]. Por outro lado, o facto da memória interna da *FPGA* operar a uma frequência inferior à *cache* possibilita um desempenho médio da *cache* tipicamente superior ao da memória interna.

É ainda importante salientar que os valores obtidos referentes a esta implementação não correspondem à situação de pior caso, podendo variar ao longo da execução do sistema. Na verdade, a memória *cache* introduz indeterminismo no sistema, motivado pelo facto do tempo de acesso a uma determinada posição de memória não ser constante e depender do estado interno da memória *cache*.

Estes tempos poderiam ser inferiores se os processadores fossem dotados do suporte adequado à comutação rápida do contexto, mais concretamente, se o banco de registos fosse constituído por diversas páginas de registos, uma associada a cada modo de operação ou conjunto de tarefas. Em caso de excepção ou na comutação de tarefas seria necessário apenas comutar de página de registos. Mais tarde quando se pretendesse regressar à tarefa interrom-

pida bastaria voltar a comutar para a respectiva página de registos.

A utilização do protocolo *SRP* para gerir os semáforos possui também a vantagem de simplificar a gestão das páginas de registos. Sempre que o número de tarefas interrompidas ultrapassar o número total de páginas de registos, a página associada à tarefa interrompida há mais tempo é aquela que, em caso de necessidade, deve ser substituída.

4.4.5.2 Processamento Periódico e Comutação de Tarefas

A tabela 4.2 mostra a variação do tempo máximo gasto no processamento periódico e na comutação de tarefas nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número máximo de tarefas do sistema.

O processamento periódico no executivo *OReK* consiste essencialmente nas seguintes operações:

- Incremento do valor do temporizador do sistema;
- Actualização das prioridades e contadores temporais das tarefas;
- Manutenção das listas de tarefas *idle*, *ready* e *preempted* (excepto na implementação *OReK-ARPA-C2*);
- Activação das tarefas periódicas de acordo com os seus parâmetros temporais e respectivos contadores;
- Activação das tarefas aperiódicas de acordo com as suas restrições temporais e pedidos de activação explícitos ou interrupções geradas;
- Detecção de violações temporais.

Nas implementações *OReK-ARPA-Sw*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* estas operações são completamente realizadas em *software*. Na implementação *OReK-ARPA-C2* são efectuadas em *hardware* no coprocessador *Cop2-OSC* em paralelo com a execução em *software* das tarefas do sistema.

A comutação de tarefas consiste na sinalização da preempção ou da terminação da tarefa actual e lançamento em execução da tarefa seguinte. Neste item não estão incluídos os procedimentos de escalonamento ou selecção da próxima tarefa a executar nem a salvaguarda e restauro do contexto da tarefa. Na comutação da tarefa considera-se apenas a transição de estado da tarefa actualmente em execução e da próxima tarefa a executar, assim como a troca da pilha activa, reflectindo desta forma a arquitectura e o funcionamento de todas as implementações realizadas. A comutação de tarefas é efectuada nas implementações realizadas inteiramente em *software* através da manipulação das listas de tarefas e na implementação *OReK-ARPA-C2* com as instruções `tprept` e `tdispt` do coprocessador *Cop2-OSC* [Arn07].

Nas implementações *OReK-ARPA-Sw*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* o processamento periódico e a comutação de tarefas são ambas realizadas pela função `TimerCallback` (ver capítulo 3) invocada no contexto da rotina de serviço à interrupção do temporizador e nas situações de terminação de uma tarefa

ou sinalização de um semáforo com decréscimo do tecto do contexto. Cada vez que ocorre uma interrupção periódica do temporizador do sistema, o contexto da tarefa actual é salvaguardado, são realizadas as operações de gestão internas do núcleo, executado o algoritmo de escalonamento para depois ser restaurado o contexto da tarefa a executar. Isto é feito mesmo que seja retomada a execução da mesma tarefa que acabou de ser interrompida.

Na implementação *OReK-ARPA-C2* a comutação de tarefas é realizada na função `OSCCallback` (ver capítulo 3) invocada no contexto da rotina de tratamento das excepções do coprocessador *Cop2-OSC* [Arn07].

Implementação	*OReK-ARPA-Sw*	*OReK-ARPA-C2*	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Tempos de Execução (μs)	12,7	1,0+(1,0)	5,4	9,7	3,3	7,3	3,0	4
	21,9	1,0+(2,0)	10,2	18,9	3,8	6,4	3,5	8
	40,2	1,0+(4,0)	19,8	37,2	4,7	13,4	4,5	16
	76,9	1,0+(8,0)	39,0	73,9	6,6	21,6	6,4	32
	150,2	1,0+(16,0)	77,4	147,2	10,5	38,4	10,4	64
	296,9	1,0+(32,0)	154,2	293,8	18,2	71,4	18,1	128

Tabela 4.2: Variação dos tempos de processamento periódico e comutação de tarefas no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas.

A tabela 4.2 mostra nas colunas relativas às implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* o tempo de execução das funções `TimerCallback` e `OSCCallback` (ver capítulo 3), respectivamente. Na coluna *OReK-ARPA-C2* é também mostrado dentro de parêntesis o tempo de processamento periódico na implementação *OReK-ARPA-C2* em função do número de tarefas. Este tempo não deve ser considerado para efeitos da carga computacional do *software* do executivo *OReK*, embora tenha que ser tomado em conta na definição do período ou resolução temporal do sistema, uma vez que o limita inferiormente.

O processamento periódico e a comutação de tarefas é um dos parâmetros avaliados em que a utilização do coprocessador *Cop2-OSC* e da implementação *OReK-ARPA-C2* leva a maiores reduções do tempo de execução e da carga computacional do *software* do executivo *OReK*. No caso de 128 tarefas são alcançados ganhos de três ordens de grandeza, quando comparado com a implementação *OReK-ARPA-Sw*. Para as mesmas 128 tarefas, no caso da implementação *OReK-PPC-Cached* foram obtidos ganhos superiores a oito vezes, relativamente à implementação *OReK-ARPA-Sw*. A implementação *OReK-MB-Int* possui tempos de processamento de aproximadamente metade em relação à implementação *OReK-ARPA-Sw*, estes devem-se principalmente à maior frequência de funcionamento do processador *MicroBlaze*. As implementações *OReK-MB-OPB*, *OReK-PPC-Int* e *OReK-PPC-DDR* possuem ganhos de ordens diferentes, motivados pelas diferenças impostas nas diversas implementações.

4.4.5.3 Selecção da Próxima Tarefa a Executar

A selecção da próxima tarefa a executar constitui parte do algoritmo de escalonamento sendo tratada separadamente, uma vez que a outra parte do escalonamento das tarefas já

está incluída no processamento periódico apresentado na secção anterior.

Este procedimento é realizado dentro da função `TimerCallback` nas implementações *OReK-ARPA-Sw*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e ainda *OReK-PPC-Cached*. O mesmo procedimento é feito completamente em *hardware* em paralelo com a execução do *OReK* ou da aplicação na implementação *OReK-ARPA-C2*.

Implementação em software - Considerando que nas versões *OReK-ARPA-Sw*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, as listas das tarefas interrompidas e das tarefas prontas a executar para os diversos níveis de criticalidade são mantidas ordenadas, a selecção da próxima tarefa a executar é trivial, consistindo na pesquisa da tarefa mais prioritária entre todas as tarefas que se encontram à cabeça dessas listas. A tarefa mais prioritária encontrada é lançada em execução, ou alternativamente a tarefa índice "0" (tarefa de entrada do sistema - *idle task*) em caso de inexistência de pelo menos uma tarefa interrompida ou ausência de uma tarefa pronta a executar.

A tabela 4.3 mostra o tempo de selecção da próxima tarefa a executar no executivo *OReK* nas implementações acima referidas, o qual não depende do número de tarefas do sistema porque para este efeito a pesquisa só é realizada à cabeça das listas.

Implementação	**Tempo de Execução** (μs)
OReK-ARPA-Sw	2,8
OReK-MB-Int	0,7
OReK-MB-OPB	1,0
OReK-PPC-Int	0,5
OReK-PPC-DDR	0,7
OReK-PPC-Cached	0,5

Tabela 4.3: Tempo de selecção da próxima tarefa a executar no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Na implementação *OReK-MB-Int* foram obtidos tempos quatro vezes menores, relativamente à implementação *OReK-ARPA-Sw*. Tal discrepância deve-se grandemente à maior frequência de funcionamento do processador *MicroBlaze*. Por outro lado, a implementação *OReK-PPC-Int* apresenta um tempo ligeiramente inferior à *OReK-MB-Int*, ou seja, o ganho obtido é menor ao que seria de esperar. Este pode ser justificado pela latência no acesso à memoria e ainda pelo conjunto de instruções do processador e diferentes optimizações introduzidas pelo compilador.

As implementações *OReK-MB-OPB*, *OReK-PPC-DDR* e *OReK-PPC-Cached* possuem ganhos de ordens diferentes, motivados pelas diferenças impostas nas diversas implementações.

Implementação em hardware - No caso da versão *OReK-ARPA-C2* do executivo *OReK* suportada pelo coprocessador *Cop2-OSC* o tempo de escalonamento das tarefas depende dos parâmetros usados na síntese do processador e da frequência de funcionamento do mesmo, a

qual está obviamente limitada superiormente pelo inverso do atraso correspondente ao caminho crítico do circuito.

A tabela 4.4 mostra o tempo de selecção da próxima tarefa a executar no executivo *OReK* na implementação *OReK-ARPA-C2* em função do número de tarefas.

Implementação	*OReK-ARPA-C2*	**Nº de Tarefas**
Tempos de Execução (μs)	0,21	4
	0,38	8
	0,71	16
	0,75	32
	0,83	64
	1,00	128

Tabela 4.4: Variação dos tempos de selecção da próxima tarefa a executar no executivo *OReK* na implementação *OReK-ARPA-C2* em função do número de tarefas.

4.4.5.4 Início de uma Instância de uma Tarefa

A tabela 4.5 mostra o tempo de início de uma instância de uma tarefa nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

A execução de uma nova instância de uma tarefa é completamente realizada em *software* através da invocação do respectivo método de entrada, pelo que não depende da implementação nem do número de tarefas do sistema.

Implementação	**Tempo de Execução (μs)**
OReK-ARPA-Sw	1,1
OReK-ARPA-C2	1,1
OReK-MB-Int	0,2
OReK-MB-OPB	0,4
OReK-PPC-Int	0,3
OReK-PPC-DDR	0,7
OReK-PPC-Cached	0,2

Tabela 4.5: Tempos de início de uma instância de uma tarefa no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Na implementação *OReK-MB-Int* foram obtidos ganhos superiores a cinco vezes, relativamente à implementação *OReK-ARPA-Sw*. Estes devem-se grandemente à maior frequência de funcionamento do processador *MicroBlaze*.

A implementação *OReK-PPC-Int* apresenta um tempo ligeiramente superior à *OReK-MB-Int*. Este pode ser justificado devido ao desempenho da implementação *OReK-PPC-Int* estar a ser penalizado pelo tempo de acesso à memoria e ainda pelo número de instruções geradas,

resultantes do conjunto de instruções do processador e diferentes optimizações introduzidas pelo compilador.

As implementações *OReK-MB-OPB* e *OReK-PPC-DDR* possuem um tempo aproximadamente duas vezes superior às *OReK-MB-Int* e *OReK-PPC-Int*, respectivamente. Esta penalização temporal deve-se maioritariamente à latência existente no acesso às respectivas memórias.

Por último, a implementação *OReK-PPC-Cached* possui o menor tempo, sendo apenas igualada pela *OReK-MB-Int*, facto que comprova a optimização de desempenho introduzida pela *cache* mas à custa de perda de determinismo.

4.4.5.5 Terminação de uma Instância de uma Tarefa

A tabela 4.6 mostra o tempo de terminação de uma instância de uma tarefa nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas.

A terminação de uma instância de uma tarefa envolve as seguintes operações:

- A sinalização da terminação da tarefa e correspondente mudança de estado;
- A verificação da existência de pedidos de paragem ou destruição pendentes e em caso afirmativo parar ou destruir a tarefa, respectivamente;
- A inserção ordenada numa das listas *idle* de acordo com os seus parâmetros temporais (excepto na implementação *OReK-ARPA-C2*) ou execução da instrução `ttermn` (na implementação *OReK-ARPA-C2*);
- A inicialização da pilha para a próxima instância da tarefa;
- A geração de uma excepção ou interrupção para que seja lançada a próxima tarefa em execução.

O tempo de terminação de uma instância de uma tarefa na implementação *OReK-ARPA-C2* é sempre inferior ao da *OReK-ARPA-Sw* e independente do número de tarefas do sistema. Este tempo varia em todas as implementações em *software* devido à inserção ordenada numa das listas *idle*. De referir que somente na situação do sistema executar com quatro tarefas e para as implementações *OReK-MB-Int*, *OReK-PPC-Int* e *OReK-PPC-Cached* obtiveram-se valores de terminação de uma instância de uma tarefa inferiores ou iguais à implementação *OReK-ARPA-C2*. Por outro lado, as implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, possuem valores inferiores relativamente à *OReK-ARPA-Sw*. De facto, os valores são consistentes com as características dos sistemas integrados implementados, sendo a frequência de funcionamento dos diversos processadores e a latência de acesso às diversas memórias os factores que mais contribuem para estes resultados.

4.4.5.6 Activação de uma Tarefa Aperiódica por uma Interrupção

A tabela 4.7 mostra o tempo de activação de uma tarefa aperiódica por uma interrupção nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*,

Implementação	*OReK-ARPA-Sw*	*OReK-ARPA-C2*	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Tempos de Execução (μs)	10,0	2,6	1,7	2,8	2,6	6,3	2,3	4
	11,7	2,6	2,2	3,7	3,0	7,8	2,7	8
	15,0	2,6	3,0	5,2	3,8	11,2	3,3	16
	21,7	2,6	4,6	8,2	5,4	17,5	4,6	32
	35,0	2,6	7,8	14,3	8,6	30,5	7,2	64
	61,7	2,6	14,1	26,3	15,0	56,3	12,3	128

Tabela 4.6: Variação dos tempos de terminação de uma instância de uma tarefa no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas.

OReK-PPC-Int, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Esta é uma operação realizada por *software* para todas as implementações com excepção da *OReK-ARPA-C2*, em que é realizada por *hardware* no *Cop2-OSC*, razão pela qual o seu tempo é nulo.

As linhas *OReK-ARPA-Sw*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* da tabela 4.7 mostram o tempo máximo de execução da função `IntCallback` (ver capítulo 3), o qual é independente do número de tarefas do sistema. Uma vez que esta função executa no contexto da rotina de serviço à interrupção é apenas necessário salvaguardar o contexto mínimo da tarefa em execução e sinalizar o pedido de activação da tarefa aperiódica correspondente à interrupção gerada.

Implementação	**Tempo de Execução (μs)**
OReK-ARPA-Sw	2,3
OReK-ARPA-C2	0,0
OReK-MB-Int	0,2
OReK-MB-OPB	0,3
OReK-PPC-Int	0,2
OReK-PPC-DDR	0,5
OReK-PPC-Cached	0,2

Tabela 4.7: Tempos de activação de uma tarefa aperiódica por uma interrupção no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.5.7 Latência Desde a Salvaguarda do Contexto até à Invocação da Função TimerCallback

A tabela 4.8 mostra a latência desde a salvaguarda do contexto da tarefa em execução, provocada por uma interrupção, até à invocação da função `TimerCallback` do executivo *OReK*, nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

É importante referir que esta latência deve-se principalmente à existência de uma rotina de serviço intermédia, inerente ao sistema desenvolvido. Resumidamente, o atendimento a interrupções por parte dos sistemas integrados *MB-SoC* e *PPC-SoC* despoleta a execução de uma primeira rotina de serviço à interrupção, responsável pela salvaguarda do contexto. De seguida, a execução passa para a rotina de serviço intermédia, do controlador de interrupções. Esta verifica por *software* em qual das entradas do controlador foi recebido o pedido de interrupção e passa o controlo de execução para a respectiva rotina. No caso da interrupção ter sido gerada pelo temporizador do sistema, a próxima rotina a executar será a função `TimerCallback`. Por último, de referir ainda que esta situação não acontece no processador *ARPA-MT* porque este não necessita de uma rotina de serviço intermédia (análoga à `TimerCallback`) para saber qual a entrada que gerou o pedido de interrupção.

Os tempos apresentados na tabela 4.8 são relativamente próximos, motivados pelas diferenças impostas nas diversas implementações.

Implementação	**Tempo de Execução (μs)**
OReK-MB-Int	0,7
OReK-MB-OPB	1,1
OReK-PPC-Int	0,9
OReK-PPC-DDR	1,7
OReK-PPC-Cached	1,3

Tabela 4.8: Latência desde a salvaguarda do contexto até a invocação da função `TimerCallback` do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.5.8 Latência Desde o Retorno da Função TimerCallback até ao Restauro do Contexto

A tabela 4.9 mostra a latência do caminho inverso, ou seja, desde o retorno da função `TimerCallback` do executivo *OReK* até ao restauro do contexto da tarefa a executar, nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Implementação	**Tempo de Execução (μs)**
OReK-MB-Int	0,7
OReK-MB-OPB	1,0
OReK-PPC-Int	1,7
OReK-PPC-DDR	3,2
OReK-PPC-Cached	1,6

Tabela 4.9: Latência desde o retorno da função `TimerCallback` até ao restauro do contexto do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Esta latência deve-se principalmente à rotina de serviço do controlador de interrupções, necessária para o correcto funcionamento do sistema. Neste caso em particular, a latência

é maioritariamente devida ao percurso feito pela rotina de serviço do controlador de interrupções até devolver a execução à rotina que efectua o restauro do contexto e ainda pelo atendimento do temporizador no sentido de limpar a interrupção gerada.

Apesar do código de alto nível (portável e em linguagem *C*) ser o mesmo, o código máquina correspondente às diversas implementações (*OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*) é necessariamente diferente. Este facto aliado aos diferentes desempenhos das arquitecturas implementadas justifica as variações temporais obtidas.

4.4.5.9 Latência Desde a Salvaguarda do Contexto até à Invocação da Função IntCallback

A tabela 4.10 mostra a latência desde a salvaguarda do contexto da tarefa em execução, provocada por uma interrupção, até à invocação da função `IntCallback` do executivo *OReK*, nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

A latência medida é análoga à secção 4.4.5.7. Neste caso, após a invocação da rotina de serviço do controlador de interrupções, a próxima rotina a executar será a função `IntCallback`, em vez da `TimerCallback`.

Novamente, apesar do código de alto nível ser o mesmo, o código máquina correspondente às diversas implementações é necessariamente diferente. Este facto aliado aos diferentes desempenhos das arquitecturas implementadas explica as variações temporais obtidas.

Por fim, o facto dos valores obtidos serem maiores que os da secção 4.8 deve-se ao facto da rotina de serviço do controlador de interrupções efectuar uma pesquisa de pedidos de interrupção activos para as diversas entradas por *software*, de forma sequencial. Como os dois sistemas integrados conectam a primeira entrada do controlador de interrupções ao temporizador, as entradas que se seguem terão sucessivamente maior latência.

Implementação	**Tempo de Execução (μs)**
OReK-MB-Int	0,8
OReK-MB-OPB	0,9
OReK-PPC-Int	1,2
OReK-PPC-DDR	1,9
OReK-PPC-Cached	1,6

Tabela 4.10: Latência desde a salvaguarda do contexto até a invocação da função `IntCallback` do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.5.10 Latência desde o Retorno da Função IntCallback até ao restauro do Contexto

A tabela 4.11 mostra a latência desde o retorno da função `IntCallback` do executivo *OReK* até ao restauro do contexto da tarefa a executar, nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

De forma análoga ao anteriormente referido esta latência deve-se, na sua maioria, ao percurso feito pela rotina de serviço do controlador de interrupções até devolver a execução à rotina que efectua o restauro do contexto. Neste caso em concreto, a latência é ainda agravada pelo atendimento do módulo *GPIO* (*General Purpose Input/Output*) que implementa um porto de entrada, no sentido de limpar a interrupção gerada.

Implementação	**Tempo de Execução (μs)**
OReK-MB-Int	1,5
OReK-MB-OPB	1,8
OReK-PPC-Int	2,2
OReK-PPC-DDR	3,5
OReK-PPC-Cached	2,1

Tabela 4.11: Latência desde o retorno da função `IntCallback` até ao restauro de contexto do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.6 Serviços Iniciados pela Aplicação

Os serviços iniciados pela aplicação cujo tempo de execução foi medido nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* do executivo *OReK* foram os seguintes:

- Inicialização do executivo;
- Leitura do temporizador do sistema;
- Activação e desactivação da preempção das tarefas;
- Criação e destruição uma tarefa;
- Arranque e paragem de uma tarefa;
- Activação explícita de uma tarefa aperiódica;
- Leitura do estado de uma tarefa;
- Criação e destruição de um semáforo;
- Registo de uma tarefa num semáforo;
- Activação de um semáforo;
- Bloqueio e libertação de um semáforo.

O tempo de execução de cada um destes serviços depende da frequência de funcionamento dos processadores *PowerPC 405*, *MicroBlaze* e *ARPA-MT* com o coprocessador *Cop2-OSC* associado. As subsecções seguintes mostram os resultados da sua avaliação temporal. Os valores apresentados consideram a execução dos serviços no interior do núcleo.

4.4.6.1 Inicialização do Executivo

As tabelas 4.12 e 4.13 mostram a variação do tempo de inicialização do executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas e de semáforos do sistema, respectivamente.

Da análise das tabelas 4.12 e 4.13 pode concluir-se que o aumento do número de tarefas e/ou de semáforos contribui para o crescimento do tempo de inicialização devido à maior quantidade de estruturas de controlo que devem ser preenchidas.

Implementação	*OReK-ARPA-Sw*	*OReK-ARPA-C2*	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Tempos de Execução (μs)	26,7	7,3	21,3	32,3	24,9	47,7	26,0	4
	39,7	7,4	22,8	36,5	26,5	54,7	28,7	8
	65,7	7,8	25,9	45,0	29,4	68,2	34,1	16
	117,7	8,4	31,8	61,9	35,6	95,5	45,0	32
	221,7	9,8	43,7	95,6	48,6	149,4	66,6	64
	429,7	12,4	67,6	163,1	74,2	258,0	110,0	128

Tabela 4.12: Variação dos tempos de inicialização do executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas (número de semáforos = 0).

O tempo de inicialização da implementação *OReK-ARPA-C2* é sempre inferior às restantes implementações devido à configuração simultânea em *hardware* e em *software* dos campos do executivo e dos blocos de controlo das tarefas e dos semáforos.

Implementação	*OReK-ARPA-Sw*	*OReK-ARPA-C2*	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Tempos de Execução (μs)	30,0	7,4	26,9	42,1	33,0	66,1	33,1	4
	46,3	7,8	29,2	48,6	34,4	76,9	37,2	8
	79,0	8,4	34,3	62,1	39,8	97,3	45,8	16
	144,3	9,8	43,8	88,3	50,1	139,2	62,7	32
	275,0	12,4	63,7	142,0	71,5	224,5	97,3	64
	536,3	17,8	101,6	246,8	111,8	390,5	164,4	128

Tabela 4.13: Variação dos tempos de inicialização do executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de semáforos (número de tarefas = número de semáforos).

No caso de 128 tarefas e 128 semáforos, a implementação *OReK-ARPA-C2* apresenta, relativamente às *OReK-ARPA-Sw*, *OReK-MB-Int* e *OReK-PPC-Int* uma redução de 30 vezes, 5 vezes e quase 6 vezes, respectivamente, do tempo de inicialização. Ainda para o mesmo caso, as implementações *OReK-MB-OPB*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, possuem tempos de inicialização menores que a implementação *OReK-ARPA-Sw*.

4.4.6.2 Leitura do Valor do Temporizador do Sistema

A tabela 4.14 mostra o tempo de leitura do temporizador do sistema nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-*

PPC-DDR e *OReK-PPC-Cached*.

Da análise da tabela 4.14 pode-se concluir que o tempo de leitura do temporizador do sistema é superior na implementação *OReK-ARPA-C2*, uma vez que deve ser lido do coprocessador *Cop2-OSC*, enquanto nas restantes implementações já se encontra num campo do *software* do núcleo do executivo. No entanto, em qualquer dos casos é inferior a um microsegundo. De referir ainda que o aumento do número de tarefas e/ou de semáforos não contribui para uma variação do tempo de leitura do temporizador do sistema.

Implementação	**Tempo de Execução** (μs)
OReK-ARPA-Sw	0,2
OReK-ARPA-C2	0,3
OReK-MB-Int	0,1
OReK-MB-OPB	0,2
OReK-PPC-Int	0,1
OReK-PPC-DDR	0,1
OReK-PPC-Cached	0,1

Tabela 4.14: Tempos de leitura do valor do temporizador do executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.6.3 Activação e Desactivação da Preempção das Tarefas

As tabelas 4.15 e 4.16 mostram o tempo de activação e de desactivação da preempção das tarefas nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, respectivamente.

Implementação	**Tempo de Execução** (μs)
OReK-ARPA-Sw	0,5
OReK-ARPA-C2	0,5
OReK-MB-Int	0,1
OReK-MB-OPB	0,1
OReK-PPC-Int	0,1
OReK-PPC-DDR	0,1
OReK-PPC-Cached	0,1

Tabela 4.15: Tempos de activação da preempção das tarefas no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Da análise das tabelas 4.15 e 4.16 pode concluir-se que estes tempos são relativamente reduzidos. Além disso, são também independentes do número e do estado das tarefas e dos semáforos, uma vez que a operação realizada nestes serviços é apenas a modificação de um indicador usado no controlo da preempção das tarefas. Este indicador é uma variável nas implementações realizadas inteiramente em *software* e um bit de um registo do coprocessador *Cop2-OSC* na implementação *OReK-ARPA-C2*.

Implementação	Tempo de Execução (μs)
OReK-ARPA-Sw	0,5
OReK-ARPA-C2	0,5
OReK-MB-Int	0,1
OReK-MB-OPB	0,1
OReK-PPC-Int	0,1
OReK-PPC-DDR	0,1
OReK-PPC-Cached	0,1

Tabela 4.16: Tempos de desactivação da preempção das tarefas no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.6.4 Criação de uma Tarefa

A tabela 4.17 mostra o tempo de criação de uma tarefa nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Os tempos mostrados correspondem à validação dos parâmetros da tarefa, à alocação de um *TCB* e à configuração dos respectivos campos nas tarefas de tempo-real críticas, uma vez que são aquelas que possuem um maior número de parâmetros associados. Os restantes tipos de tarefas possuem tempos de criação ligeiramente inferiores. No entanto, qualquer que seja o seu tipo, o tempo de criação de uma tarefa não depende do número de tarefas do sistema nem dos seus estados particulares.

Implementação	Tempo de Execução (μs)
OReK-ARPA-Sw	6,2
OReK-ARPA-C2	5,5
OReK-MB-Int	2,4
OReK-MB-OPB	5,6
OReK-PPC-Int	1,8
OReK-PPC-DDR	4,2
OReK-PPC-Cached	1,8

Tabela 4.17: Tempos de criação de uma tarefa no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

O tempo na implementação *OReK-ARPA-C2* é ligeiramente inferior ao da *OReK-ARPA-Sw* porque as instruções `tcnort`, `tcsrtp`, `tcsrta`, `tchrtp` e `tchrta` disponibilizadas pelo *Cop2-OSC* para este efeito realizam a alocação do *TCB* e a inicialização do respectivo registo de estado de forma atómica num único ciclo de relógio [Arn07]. Os restantes campos do *TCB* são preenchidos por *software* de forma sequencial em todas as implementações. Nas implementações *OReK-MB-Int*, *OReK-PPC-Int* e *OReK-PPC-Cached*, o tempo de criação de uma tarefa é aproximadamente 3 vezes inferior ao da implementação *OReK-ARPA-Sw*.

4.4.6.5 Destruição de uma Tarefa

A tabela 4.18 mostra o tempo de destruição de uma tarefa nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Os tempos mostrados correspondem à validação do identificador da tarefa, à verificação se a tarefa pode (ou não) ser imediatamente destruída e à destruição propriamente ou ao seu deferimento. Qualquer que seja o seu tipo, o tempo de destruição de uma tarefa não depende do número de tarefas do sistema nem dos seus estados particulares.

Implementação	**Tempo de Execução (μs)**
OReK-ARPA-Sw	7,7
OReK-ARPA-C2	3,3
OReK-MB-Int	2,4
OReK-MB-OPB	5,0
OReK-PPC-Int	3,2
OReK-PPC-DDR	7,9
OReK-PPC-Cached	3,0

Tabela 4.18: Tempos de destruição de uma tarefa no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

O tempo na implementação *OReK-ARPA-C2* é cerca de metade da *OReK-ARPA-Sw* porque a instrução `tdty` disponibilizada pelo *Cop2-OSC* para este efeito realiza a libertação do *TCB* num único ciclo de relógio e, no caso das tarefas associadas a fontes de interrupção, a desactivação da respectiva fonte [Arn07]. À excepção da implementação *OReK-PPC-DDR*, as restantes implementações possuem tempos de destruição de uma tarefa inferiores à *OReK-ARPA-Sw*. Na verdade, as implementações *OReK-MB-OPB* e *OReK-PPC-DDR*, possuem tempos consideravelmente superiores em relação às *OReK-MB-Int*, *OReK-PPC-Int* e *OReK-PPC-Cached*. Tal deve-se maioritariamente à penalização temporal introduzida por memórias com acesso mais lento.

4.4.6.6 Arranque de uma Tarefa

A tabela 4.19 mostra a variação do tempo máximo de arranque de uma tarefa nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas do sistema.

No arranque de uma tarefa o tempo é gasto na validação do identificador da tarefa, na verificação do seu estado, na alocação e inicialização da pilha e na transição da tarefa para o estado *idle* ou *ready* consoante o seu tipo e parâmetros iniciais.

À excepção da implementação *OReK-ARPA-C2*, o tempo cresce com o aumento do número de tarefas porque as tarefas no arranque são colocadas numas das listas de tarefas *idle* ou *ready*, as quais são ordenadas por tempo para a activação e prioridade, respectivamente. O

tempo máximo corresponde à situação mais desfavorável, em que a colocação de uma nova tarefa numa dessas listas requer o seu atravessamento completo.

Na implementação *OReK-ARPA-C2* o tempo de arranque de uma tarefa é constante e inferior ao das restantes implementações porque a instrução `tstart` disponibilizada pelo *Cop2-OSC* para este efeito realiza a verificação e a transição do estado da tarefa num único ciclo de relógio [Arn07]. No caso das tarefas associadas a fontes de interrupção é também feita a associação com a respectiva fonte. Na implementação *OReK-ARPA-C2*, como não existem listas de tarefas, todas as operações demoram um tempo fixo a executar.

Uma tarefa depois de arrancada e activada será considerada pelo escalonador de tarefas. Tal como já foi explicado, na implementação *OReK-ARPA-C2*, o número máximo de tarefas afecta o tempo de escalonamento das mesmas, não as transições de estado.

As implementações *OReK-MB-Int* e *OReK-PPC-Int* possuem tempos semelhantes. Estas, comparativamente com a implementação *OReK-ARPA-Sw*, apresentam um tempo aproximadamente duas vezes inferior. Os maiores tempos obtidos estão associados à implementação *OReK-PPC-DDR*, sendo 20 vezes superior ao da implementação *OReK-ARPA-C2* com 128 tarefas.

Implementação	*OReK-ARPA-Sw*	*OReK-ARPA-C2*	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Tempos de Execução (μs)	10,9	4,3	4,5	7,9	4,7	13,6	5,0	4
	12,6	4,3	5,1	8,9	5,2	15,9	5,4	8
	15,9	4,3	6,2	11,1	6,3	21,0	6,4	16
	22,6	4,3	8,4	15,5	8,6	30,8	8,3	32
	35,9	4,3	12,7	24,2	13,1	50,4	12,3	64
	62,6	4,3	21,7	42,1	22,0	89,0	20,7	128

Tabela 4.19: Variação do tempo de arranque de uma tarefa no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas.

4.4.6.7 Paragem de uma Tarefa

A tabela 4.20 mostra o tempo de paragem de uma tarefa nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Os tempos mostrados correspondem à validação do identificador da tarefa, à verificação se a tarefa pode (ou não) ser imediatamente parada e à paragem propriamente ou ao seu deferimento. Qualquer que seja o seu tipo, o tempo de paragem de uma tarefa não depende do número de tarefas do sistema nem dos seus estados particulares.

O tempo na implementação *OReK-ARPA-C2* é cerca de metade da *OReK-ARPA-Sw* porque a instrução `tstop` disponibilizada pelo *Cop2-OSC* para este efeito realiza a transição de estado do *TCB* num único ciclo de relógio e, no caso das tarefas associadas a fontes de interrupção, a desactivação da respectiva fonte [Arn07]. As implementações *OReK-MB-Int* e *OReK-PPC-Int*, analogamente ao arranque de uma tarefa, possuem tempos duas a três vezes inferiores à implementação *OReK-ARPA-Sw*. À excepção da implementação *OReK-PPC-Cached* que, como referido anteriormente, não nos permite obter a situação do pior caso, as restantes implementações (*OReK-MB-OPB* e *OReK-PPC-DDR*) possuem tempos consideravelmente maiores, proporcionados pelas abordagens de implementação utilizadas.

Implementação	Tempo de Execução (μs)
OReK-ARPA-Sw	7,5
OReK-ARPA-C2	3,1
OReK-MB-Int	2,4
OReK-MB-OPB	4,3
OReK-PPC-Int	3,0
OReK-PPC-DDR	8,1
OReK-PPC-Cached	4,0

Tabela 4.20: Tempos de paragem de uma tarefa no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.6.8 Activação Explícita de uma Tarefa Aperiódica

A tabela 4.21 mostra a variação do tempo máximo de activação explícita de uma tarefa aperiódica nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas do sistema.

Na activação de uma tarefa, o tempo é gasto na validação do seu identificador, na verificação do seu tipo e estado actual e na transição imediata ou deferida da tarefa para o estado *ready*, consoante o atraso especificado.

Implementação	*OReK-ARPA-Sw*	*OReK-ARPA-C2*	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Tempos de Execução (μs)	7,5	2,2	2,1	3,7	2,0	5,2	1,8	4
	9,2	2,2	2,7	5,0	2,5	7,7	2,3	8
	12,5	2,2	4,0	7,3	3,6	12,6	3,2	16
	19,2	2,2	6,6	12,1	5,9	22,2	5,1	32
	32,5	2,2	11,9	21,8	10,4	41,7	9,0	64
	59,2	2,2	22,4	40,9	19,3	80,6	16,8	128

Tabela 4.21: Variação do tempo de activação explícita de uma tarefa aperiódica no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas.

À excepção da implementação *OReK-ARPA-C2*, o tempo cresce com o aumento do número de tarefas porque estas, na activação, são colocadas numa das listas de tarefas *ready*, as quais são ordenadas por prioridade. O tempo máximo corresponde à situação mais desfavorável, em que a colocação de uma nova tarefa na lista requer o seu atravessamento completo.

Na implementação *OReK-ARPA-C2* o tempo de arranque de uma tarefa é inferior ao das restantes implementações com excepção para os casos em que o sistema possui 4 tarefas e nas implementações *OReK-MB-Int*, *OReK-PPC-Int* e *OReK-PPC-Cached*.

O tempo de arranque na implementação *OReK-ARPA-C2* é ainda constante devido à instrução `tactiv` disponibilizada pelo *Cop2-OSC* para este efeito [Arn07]. Esta realiza a verificação e a transição do estado da tarefa num único ciclo de relógio. Na implementação *OReK-ARPA-C2*, como não existem listas de tarefas, todas as operações demoram um tempo fixo a executar.

As implementações *OReK-MB-Int* e *OReK-PPC-Int* possuem tempos idênticos. Estas, comparativamente com a implementação *OReK-ARPA-Sw*, apresentam um tempo aproximadamente duas vezes inferior. Os maiores tempos obtidos estão associados à implementação *OReK-PPC-DDR*, sendo quase 37 vezes superior ao da implementação *OReK-ARPA-C2* e aproximadamente 5 vezes maior que o da *OReK-PPC-Int*, ambas com 128 tarefas.

De referir ainda que uma tarefa aperiódica depois de activada será considerada pelo escalonador de tarefas.

4.4.6.9 Leitura do Estado de uma Tarefa

A tabela 4.22 mostra o tempo de leitura do estado de uma tarefa nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

O tempo gasto corresponde validação do identificador, à leitura e à devolução do estado, o qual corresponde à lista em que a tarefa se encontra nas implementações em *software* ou a um campo do respectivo *TCB* do coprocessador *Cop2-OSC* na implementação *OReK-ARPA-C2*. Os tempos mostrados na tabela 4.22 não variam muito entre si, sendo o tempo menor pertencente à implementação *OReK-PPC-Int* que é aproximadamente três vezes inferior à *OReK-ARPA-Sw*.

Implementação	**Tempo de Execução** (μs)
OReK-ARPA-Sw	2,3
OReK-ARPA-C2	2,2
OReK-MB-Int	1,1
OReK-MB-OPB	1,9
OReK-PPC-Int	0,7
OReK-PPC-DDR	1,5
OReK-PPC-Cached	1,0

Tabela 4.22: Tempos de leitura do estado de uma tarefa no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.6.10 Criação de um Semáforo

A tabela 4.23 mostra o tempo de criação de um semáforo nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Os tempos mostrados correspondem à alocação de um *SCB* e à configuração do seu estado e do respectivo tecto inicial do recurso. O tempo de criação de um semáforo não depende do número de semáforos do sistema nem dos seus estados particulares.

O tempo na implementação *OReK-ARPA-C2* é ligeiramente inferior ao da *OReK-ARPA-Sw* porque a instrução `scrt` disponibilizada pelo *Cop2-OSC* para este efeito realiza a alocação do *SCB* e inicialização do respectivo registo de estado de forma atómica num único ciclo de relógio.

Mais uma vez, os tempos apresentados não variam muito entre si, sendo o tempo menor pertencente às implementações *OReK-PPC-Int* e *OReK-PPC-Cached*, este é aproximadamente duas vezes inferior à *OReK-ARPA-Sw*.

Implementação	**Tempo de Execução** (μs)
OReK-ARPA-Sw	1,7
OReK-ARPA-C2	1,5
OReK-MB-Int	1,1
OReK-MB-OPB	2,1
OReK-PPC-Int	0,8
OReK-PPC-DDR	1,5
OReK-PPC-Cached	0,8

Tabela 4.23: Tempos de criação de um semáforo no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.6.11 Destruição de um Semáforo

A tabela 4.24 mostra o tempo de destruição de um semáforo nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Os tempos mostrados correspondem à validação do identificador do semáforo, à verificação se o semáforo pode (ou não) ser destruído e à destruição propriamente dita. O tempo de destruição de um semáforo não depende do número de semáforos do sistema nem dos seus estados particulares.

Implementação	**Tempo de Execução** (μs)
OReK-ARPA-Sw	2,4
OReK-ARPA-C2	1,9
OReK-MB-Int	1,0
OReK-MB-OPB	1,9
OReK-PPC-Int	0,8
OReK-PPC-DDR	1,6
OReK-PPC-Cached	1,0

Tabela 4.24: Tempos de destruição de um semáforo no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

O tempo na implementação *OReK-ARPA-C2* é ligeiramente inferior ao da *OReK-ARPA-Sw* porque a instrução `sdty` disponibilizada pelo *Cop2-OSC* para este efeito realiza a libertação do *SCB* num único ciclo de relógio [Arn07].

Todas as restantes implementações possuem tempos inferiores à *OReK-ARPA-Sw* e o menor tempo, obtido na implementação *OReK-PPC-Int*, é três vezes inferior ao da *OReK-ARPA-Sw*, possuindo ainda uma diferença mínima relativamente à *OReK-MB-Int* e à *OReK-PPC-Cached*.

4.4.6.12 Registo de uma Tarefa num Semáforo

A tabela 4.25 mostra o tempo de registo de uma tarefa num semáforo nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Os tempos mostrados correspondem à validação dos identificadores da tarefa e do semáforo, à comparação do nível de preempção da tarefa com o tecto actual do semáforo e à sua actualização (caso seja necessário). O tempo de registo de uma tarefa num semáforo não depende do número de semáforos do sistema nem dos seus estados particulares.

O tempo na implementação *OReK-ARPA-C2* é ligeiramente superior ao da *OReK-ARPA-Sw* devido à grande maioria do trabalho ser realizada em *software*, havendo no primeiro caso o custo adicional da comunicação com o coprocessador *Cop2-OSC*.

Relativamente às restantes implementações, todas possuem tempos inferiores à *OReK-ARPA-Sw* e o menor tempo, obtido na implementação *OReK-PPC-Cached*, é aproximadamente quatro vezes inferior ao da *OReK-ARPA-C2*, possuindo ainda uma diferença mínima relativamente às implementações *OReK-MB-Int* e *OReK-PPC-Int* .

Implementação	**Tempo de Execução** (μs)
OReK-ARPA-Sw	3,5
OReK-ARPA-C2	3,8
OReK-MB-Int	1,3
OReK-MB-OPB	2,6
OReK-PPC-Int	1,1
OReK-PPC-DDR	3,2
OReK-PPC-Cached	1,0

Tabela 4.25: Tempos de registo de uma tarefa num semáforo no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.6.13 Activação de um Semáforo

A tabela 4.26 mostra o tempo de activação de um semáforo nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Os tempos mostrados correspondem à validação do identificador do semáforo, à verificação do seu estado e à sua activação propriamente dita. O tempo de activação de um semáforo não depende do número de semáforos do sistema nem dos seus estados particulares.

O tempo na implementação *OReK-ARPA-Sw* é ligeiramente inferior ao da *OReK-ARPA-C2* porque, enquanto que nas implementações em *software*, como no primeiro caso, apenas é alterado um indicador do respectivo *SCB*, no segundo caso existe um custo associado ao acesso ao coprocessador *Cop2-OSC* devido à execução da instrução `senable` [Arn07].

À excepção das implementações *OReK-MB-OPB* e *OReK-PPC-DDR*, todas as restantes possuem tempos inferiores à *OReK-ARPA-Sw* e o menor tempo, obtido na implementação *OReK-PPC-Cached*, é relativamente próximo do obtido da *OReK-ARPA-Sw*, possuindo ainda uma diferença mínima relativamente às implementações *OReK-MB-Int* e *OReK-PPC-Cached*.

Implementação	Tempo de Execução (μs)
OReK-ARPA-Sw	1,0
OReK-ARPA-C2	1,3
OReK-MB-Int	0,9
OReK-MB-OPB	1,5
OReK-PPC-Int	0,7
OReK-PPC-DDR	1,2
OReK-PPC-Cached	0,6

Tabela 4.26: Tempos de activação de um semáforo no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

4.4.6.14 Bloqueio de um Semáforo

A tabela 4.27 mostra o tempo de bloqueio de um semáforo nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Os tempos mostrados correspondem à validação do identificador do semáforo, à verificação do seu estado e ao seu bloqueio propriamente dito. O tempo de bloqueio de um semáforo não depende do número de semáforos do sistema nem dos seus estados particulares.

Implementação	Tempo de Execução (μs)
OReK-ARPA-Sw	2,6
OReK-ARPA-C2	2,1
OReK-MB-Int	1,0
OReK-MB-OPB	1,7
OReK-PPC-Int	0,9
OReK-PPC-DDR	1,8
OReK-PPC-Cached	0,8

Tabela 4.27: Tempos de bloqueio de um semáforo no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

O tempo na implementação *OReK-ARPA-C2* é ligeiramente inferior ao da *OReK-ARPA-Sw* porque a instrução `swait` disponibilizada pelo *Cop2-OSC* para este efeito realiza a verificação e a transição do estado do *SCB* num único ciclo de relógio [Arn07]. Por outro lado, as restantes implementações baseadas em processadores que permitem frequências mais elevadas, possibilitam maior capacidade de processamento, o que se traduz em tempos de operação de execução menores.

4.4.6.15 Libertação de um Semáforo

A tabela 4.28 mostra o tempo de libertação de um semáforo nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Implementação	Tempo de Execução (μs)
OReK-ARPA-Sw	4,0
OReK-ARPA-C2	2,0
OReK-MB-Int	1,1
OReK-MB-OPB	1,8
OReK-PPC-Int	0,8
OReK-PPC-DDR	1,5
OReK-PPC-Cached	0,7

Tabela 4.28: Tempos de libertação de um semáforo no executivo *OReK* nas implementações *OReK-ARPA-Sw*, *OReK-ARPA-C2*, *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*.

Os tempos mostrados correspondem à validação do identificador do semáforo, à verificação do seu estado e à sua libertação propriamente dita. O tempo de libertação de um semáforo não depende do número de semáforos do sistema nem dos seus estados particulares.

O tempo na implementação *OReK-ARPA-C2* é metade da *OReK-ARPA-Sw* porque a instrução `ssignal` disponibilizada pelo *Cop2-OSC* para este efeito realiza a verificação e a transição do estado do *SCB* num único ciclo de relógio [Arn07].

Todas as restantes implementações possuem tempos inferiores à *OReK-ARPA-C2* e o menor tempo, obtido na implementação *OReK-PPC-Cached*, é aproximadamente 6 vezes inferior ao da *OReK-ARPA-Sw*, possuindo ainda uma diferença reduzida relativamente às implementações *OReK-MB-Int* e *OReK-PPC-Cached*.

De notar que os tempos das implementações mostrados na tabela 4.28 não incluem a interrupção ou a excepção gerada em caso de decréscimo do tecto do sistema. Esta excepção é sempre gerada nas implementações por *software* para reavaliação do critério de preempção do protocolo *SRP*. No caso da *OReK-ARPA-C2*, após a libertação de um semáforo, apenas é gerada uma excepção se houver necessidade efectiva de comutar a tarefa em execução.

4.5 Avaliação da Carga Periódica Relativa

A avaliação da carga periódica relativa do conjunto formado pelo executivo *OReK* com a aplicação visa obter valores concretos da carga periódica relativa do sistema para um conjunto de implementações com diversos valores de resolução temporal e número de tarefas.

A avaliação será feita com os sistemas integrados *MB-SoC* e *PPC-SoC*. Os processadores *Microblaze* e *Power PC 405* possuem os parâmetros de configuração utilizados anteriormente na secção 4.4, com excepção da frequência de funcionamento do processador *Microblaze*. Após a medição dos valores relativos à avaliação temporal das implementações *OReK-MB-Int* e *OReK-MB-OPB*, apresentadas na secção 4.4, foi proposta uma terceira implementação baseada na plataforma *MB-SoC*. Esta é idêntica à segunda abordagem, porém utiliza uma memória externa do tipo *DDR* para armazenar *heap* e *stack*, designando-se por *OReK-MB-DDR*. A reformulação do sistema integrado *MB-SoC* para passar a suportar uma interface com a memória *DDR* impôs uma nova frequência de funcionamento do sistema. Assim sendo,

todo o sistema (incluindo o processador *MicroBlaze*) passou a operar à frequência de *100 MHz*.

Tendo em vista a avaliação da carga periódica do sistema formado pelo conjunto do executivo *OReK* e da aplicação, de forma metódica e rigorosa, foram definidos os seguintes critérios:

- O sistema deve possuir quatro temporizadores em funcionamento simultâneo, sendo três responsáveis por contabilizar as componentes temporais relativas à execução do núcleo ou executivo *OReK* (T_{OReK}), das tarefas da aplicação (T_{Appl}) e do tempo livre (T_{Idle}), respectivamente. O quarto temporizador deve ser activado no momento de arranque das tarefas, funcionando como um controlo complementar (T_{Ctrl}). Desta forma, a soma das três componentes anteriores deve resultar num valor muito próximo (idealmente igual) ao do contador de controlo;

- Para facilitar a avaliação dos parâmetros que se pretende avaliar as tarefas devem ser sintéticas, isto é, não existe interacção com o mundo exterior e o tempo de processamento que lhes está associado é definido através de um ciclo cujo número de iterações define a carga computacional absoluta da tarefa que, para além de activar o contador T_{Appl}, dissipa o tempo de processamento da tarefa. No instante de terminação da tarefa, esta deve parar o respectivo contador e activar o contador T_{OReK}, visto que a execução vai ser devolvida ao núcleo ou executivo *OReK*. Além disso, é importante que as tarefas ocupem uma carga relativa constante, o que facilita a comparação entre as diversas implementações, para as várias medições efectuadas;

- O tempo livre deve ser implementado sob a forma de um ciclo que, para além de ocupar tempo de processamento, activa o contador T_{Idle};

- No início da rotina que implementa a comutação do contexto do executivo, devem ser parados de forma simultânea, os contadores T_{Appl} e T_{Idle} e activado T_{OReK};

- No final da rotina que implementa a salvaguarda do contexto do executivo *OReK*, o temporizador T_{OReK} deve ser parado;

- O sistema deverá suportar conjuntos de tarefas com dimensão compreendida entre 16 e 128 tarefas e ser exequível para as resoluções temporais de 10 ms, 1 ms e 0,1 ms, isto é, para o pior caso a soma das componentes de cargas periódicas relativas do executivo e das tarefas tem de ser inferior, ou quando muito igual, a 100 %. De referir ainda que o pior caso deve ocorrer para a resolução temporal de 0,1 ms e 128 tarefas, que corresponde à situação de maior carga periódica relativa do executivo *OReK*;

- Para garantir uma boa precisão das diversas componentes temporais a medir, o executivo deverá funcionar durante 100000 *ticks*. Este é um valor consideravelmente grande, quando comparado com as características temporais do sistema a avaliar. Adicionalmente, foram ainda efectuados testes experimentais onde foi verificada uma precisão superior a duas casas decimais nos valores obtidos. O funcionamento do executivo *OReK* durante 100000 *ticks*, em termos absolutos, corresponde a tempos de execução que variam entre 1000 s, 100 s e 10 s, consoante a resolução temporal seja 10 ms, 1 ms e 0,1 ms, respectivamente;

- Por fim, as tarefas a implementar serão do tipo tempo-real críticas. Estas são as tarefas que exigem mais processamento por parte do executivo *OReK*, devido à necessidade de ser calculada a sua prioridade em cada instante de execução das funções de gestão do executivo *OReK* (prioridade dinâmica) e ainda devido à verificação de perda de *deadline*.

4.5.1 Carga Relativa com Período de 0,1 *m*s

Além dos pressupostos acima referidos e no sentido de manter a carga periódica relativa constante, as tarefas a implementar devem ainda possuir as seguintes características (ver figura 4.2):

- A configuração inicial consiste num conjunto mínimo formado por 16 tarefas, com fases iniciais compreendidas entre 0 e 1500, uniformemente espaçadas de 100 unidades temporais e com um período igual à *deadline* de 1600 unidades temporais (*ticks*);

- As especificações anteriores mantêm-se para implementações múltiplas do conjunto mínimo, isto é, para 32, 64 e 128 tarefas, em que são simultaneamente activadas 2, 4 ou 8 tarefas, uniformemente espaçadas de 100 unidades temporais;

- Para cada tarefa do conjunto mínimo, é especificado um majorante exequível de carga computacional, de forma a que o conjunto das tarefas seja escalonável;

- A especificação da carga computacional anterior é sucessivamente dividida por dois quando a dimensão do conjunto de tarefas duplica, isto é, para 32 tarefas a carga computacional de cada tarefa é metade da especificada para 16 tarefas, e assim sucessivamente, de forma a manter a carga global, quer em termos relativos quer em termos absolutos (obviamente que entre plataformas com processadores de diferentes desempenhos o tempo de processamento será diferente e consequentemente as cargas computacionais relativa e absoluta também o serão);

- Por questões de simplicidade de implementação todas as tarefas devem ser criadas a partir de uma única classe, isto é, são instâncias com parâmetros temporais específicos de uma classe que define o esqueleto da tarefa sintética;

- Por último, as especificações anteriores são ainda válidas para ambos os sistemas integrados *MB-SoC* e *PPC-SoC*, ou seja, as características temporais e de carga para cada tarefa são as mesmas para os dois sistemas integrados.

A figura 4.2 ilustra, de forma simplificada, a execução da aplicação constituída por um conjunto de tarefas, desenvolvida para possibilitar a avaliação da carga periódica relativa. Nesta figura são essencialmente focadas a execução do conjunto mínimo composto por 16 tarefas e a execução de 32 tarefas, todavia esta é escalável às restantes execuções de 64 e 128 tarefas.

No primeiro caso a aplicação é composta pelo conjunto mínimo de 16 tarefas. Estas estão espaçadas de 10 *ms*, o que vai de acordo com as especificações acima definidas, mais concretamente, para uma resolução de 0,1 *ms* as tarefas encontram-se uniformemente espaçadas de 100 unidades temporais. No segundo caso a aplicação é composta por 32 tarefas, sendo simultaneamente activadas 2 tarefas, de acordo com o segundo ponto das especificações acima

enunciadas. No sentido de manter a carga computacional global da aplicação, o número de tarefas duplica e a carga computacional de cada tarefa é reduzida para metade, indo novamente de acordo com as especificações previamente definidas.

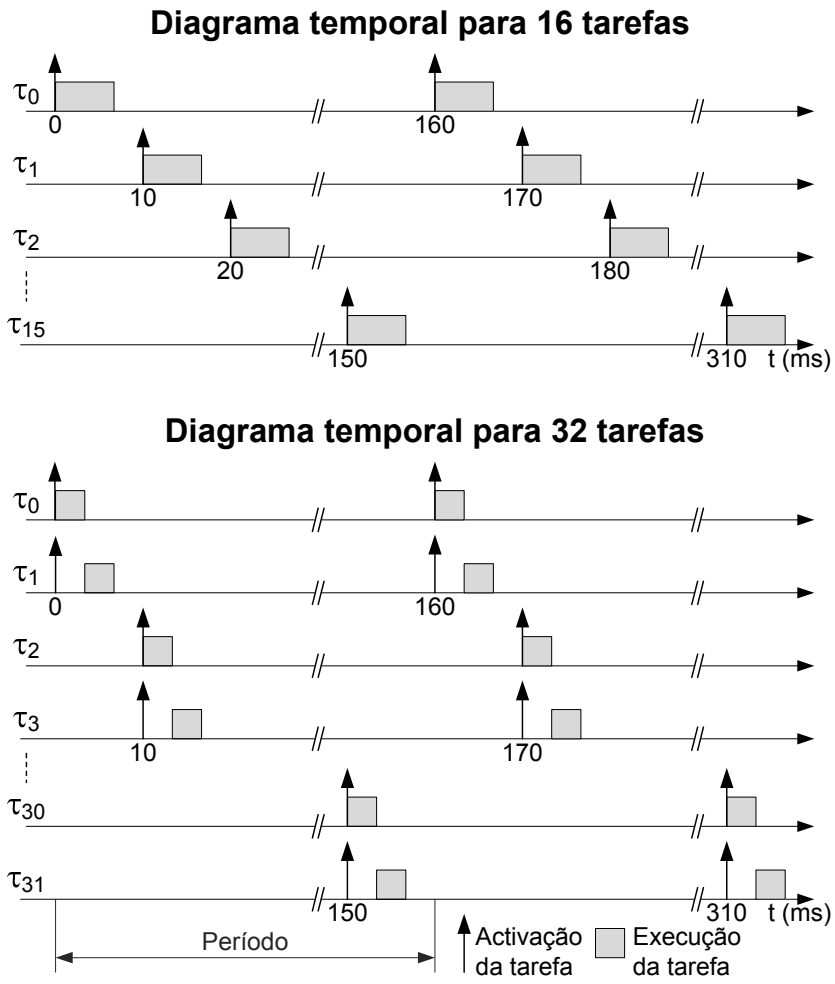


Figura 4.2: Esquema da execução do conjunto de tarefas (*task set*) desenvolvido para possibilitar a avaliação da carga periódica relativa, com 16 tarefas no primeiro caso e 32 tarefas no segundo.

4.5.1.1 Carga Relativa do Executivo OReK

A tabela 4.29 mostra a carga periódica relativa do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas e assumindo um período de 0,1 milisegundos.

As taxas mostradas correspondem à medição da execução do núcleo ou executivo *OReK* nas diversas implementações. É importante frisar que esta implementação é efectuada com tarefas sintéticas e que os resultados obtidos não podem ser classificados como majorantes

absolutos, ou seja, alterações às características das tarefas podem introduzir variações na carga relativa do executivo.

De referir ainda que o primeiro sistema integrado utilizado na avaliação da carga periódica relativa foi o *MB-SoC* descrito na secção 4.4, ou seja, sem a implementação *OReK-MB-DDR*. Por outro lado, o valor de carga temporal estipulado para as tarefas foi maximizado de forma a impor, para o pior caso, uma carga periódica relativa livre muito próxima de zero, motivado pelo facto do escalonamento *EDF* possibilitar taxas de ocupação da *CPU* de 100 %, como referido na secção 2.1.2.2.

Em todas as implementações, a carga relativa do executivo aumenta com o número de tarefas, o que se explica pela necessidade de percorrer maiores listas de tarefas para gestão interna do núcleo.

Como consequência desta abordagem, após a introdução da implementação *OReK-MB-DDR*, o sistema passou a ter uma nova situação de pior caso, motivada pela maior latência no acesso ao conteúdo da memória *DDR*. Como já haviam sido realizadas diversas medições, optou-se por manter a carga temporal das tarefas. Por esta razão, o sistema deixou de ser escalonável na implementação *OReK-MB-DDR* a partir de um número de tarefas compreendido entre 32 e 64. Este facto impossibilita a recolha de valores para 64 e 128 tarefas na respectiva implementação. O mesmo se aplica à implementação *OReK-PPC-DDR* com 128 tarefas.

Implementação	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-MB-DDR*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Carga	6,4%	12,9%	27,7%	7,7%	20,6%	7,0%	16
Relativa do	8,4%	17,3%	37,7%	9,7%	30,0%	8,6%	32
Executivo	12,4%	26,1%	-	13,8%	48,9%	11,9%	64
($T_{Tick} = 0,1ms$)	20,6%	44,1%	-	22,0%	-	19,0%	128

Tabela 4.29: Carga periódica relativa do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas e assumindo um período de 0,1 milisegundos.

Por comparação directa de valores presentes na tabela 4.29, constata-se que a carga relativa do executivo *OReK* na implementação *OReK-PPC-Int* é ligeiramente superior à análoga *OReK-MB-Int*. Esta constatação, apesar de algo inesperada, deve-se a dois factores centrais: Em primeiro lugar e como anteriormente referido, a memória interna opera à frequência de 100 *MHz* em ambas as implementações, o que penaliza temporalmente a implementação *OReK-PPC-Int*. O segundo factor está relacionado com as optimizações da compilação para os dois sistemas integrados desenvolvidos. Na verdade, a grande maioria do executivo *OReK* foi desenvolvida em linguagem de alto nível, portável. Esta é convertida pelo compilador em código máquina a executar pelo respectivo processador. Na secção 3.5.2, referente aos requisitos de memória do executivo *OReK* nas diversas implementações, pode constatar-se que a dimensão da implementação o processador *PowerPC 405* é aproximadamente 60% superior à do processador *MicroBlaze*, o que significa que o processador *PowerPC 405* vai sofrer uma sobrecarga computacional consideravelmente superior à do *MicroBlaze*, para o mesmo trabalho útil.

À excepção da *OReK-PPC-Cached*, todas as restantes implementações possuem tempos superiores à *OReK-MB-Int*, proporcionados pela maior latência ao acesso das respectivas memórias.

4.5.1.2 Tempo Livre Relativo

A tabela 4.30 mostra a carga periódica relativa livre nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas e assumindo um período de 0,1 milisegundos.

A carga relativa livre deve-se ao tempo de folga (livre) do processador na execução do sistema formado pelo conjunto das tarefas com o executivo *OReK*. Dito de outra forma, o processador só vai executar o código correspondente ao tempo livre, quando não for requerida a execução de código das tarefas ou do executivo.

Implementação	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-MB-DDR*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Tempo	38,2%	31,7%	16,9%	62,4%	45,9%	63,4%	16
Livre	36,2%	27,3%	7,1%	60,3%	36,2%	61,8%	32
Relativo	32,2%	18,5%	-	56,2%	16,6%	58,6%	64
($T_{Tick} = 0,1ms$)	24,0%	0,5%	-	47,9%	-	51,3%	128

Tabela 4.30: Carga periódica relativa livre nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas e assumindo um período de 0,1 milisegundos.

De referir que o majorante especificado para a carga temporal das tarefas permitiu obter um tempo livre mínimo de 0,5% referente à implementação *OReK-MB-OPB*, como se pode ver na tabela 4.30.

Por último, a ausência de valores nas implementações *OReK-MB-DDR* e *OReK-PPC-DDR*, para números mais elevados de tarefas, deve-se às razões apresentadas anteriormente, na secção 4.5.1.1. Caso existissem, estes representariam as implementações de pior caso.

4.5.1.3 Carga Relativa das Tarefas

A tabela 4.31 mostra a carga periódica relativa das tarefas nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas e assumindo um período de 0,1 milisegundos.

Por comparação dos valores apresentados na tabela 4.31, é imediato constatar que a carga relativa das tarefas nas implementações com o sistema integrado *PPC-SoC* é consideravelmente inferior às implementações com o *MB-SoC*. Como a carga temporal de cada tarefa é igual em ambas implementações, pode-se afirmar que o processador *PowerPC 405* possui um desempenho superior ao *MicroBlaze* na execução das tarefas em avaliação.

Implementação	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-MB-DDR*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Carga	55,4%	55,4%	55,4%	30,0%	33,6%	29,6%	16
Relativa das	55,4%	55,4%	55,2%	30,0%	33,9%	29,6%	32
Tarefas	55,4%	55,4%	-	30,0%	34,6%	29,4%	64
($T_{Tick} = 0,1ms$)	55,4%	55,4%	-	30,0%	-	29,7%	128

Tabela 4.31: Carga periódica relativa das tarefas nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas e assumindo um período de 0,1 milisegundos.

É ainda importante referir que o módulo de temporizadores, com o qual se efectuaram as medições temporais, dependendo do processador, pode ser utilizado com uma de duas interfaces diferentes. Na implementação com o processador *MicroBlaze*, a ligação ao processador com menor latência disponibilizada é a *FSL* (*Fast Simplex Link*), pelo que o módulo foi implementado com esta interface. Por outro lado, o barramento utilizado para comunicação com o processador *PowerPC 405* é o *PLB*, o que motivou a implementação de uma interface *PLB* no módulo de temporizadores.

Relativamente ao módulo de temporizadores, foram disponibilizadas as mesmas funcionalidades para ambas as versões. Porém, o sistema integrado *PPC-SoC* interliga no mesmo barramento *PLB* diversos periféricos, o módulo de interface com a memória *DDR* e o módulo de temporizadores. Este facto juntamente com uma carga relativa de tarefas elevada e um baixo período do sistema são factores que contribuem para as pequenas variações da carga relativa das tarefas, obtidas nas implementações *OReK-PPC-DDR* e *OReK-PPC-Cached*. De forma análoga se justificam as pequenas variações obtidas na implementação *OReK-MB-DDR*.

Por último, a ausência de valores nas implementações *OReK-MB-DDR* e *OReK-PPC-DDR*, para números mais elevados de tarefas, deve-se às razões apresentadas anteriormente, na secção 4.5.1.1.

4.5.2 Carga Relativa com Período de 1 *ms*

Além aos pressupostos referidos na secção 4.5 e no sentido de manter a carga periódica relativa constante, as tarefas a implementar devem ainda possuir as seguintes características temporais (ver figura 4.2):

- A configuração inicial consiste num conjunto mínimo formado por 16 tarefas, com fases iniciais compreendidas entre 0 e 150, uniformemente espaçadas de 10 unidades temporais e com um período igual à *deadline* de 160 unidades temporais (*ticks*);
- As especificações anteriores mantêm-se para implementações múltiplas do conjunto mínimo, isto é, para 32, 64 e 128 tarefas, em que são simultaneamente activadas 2, 4 ou 8 tarefas, uniformemente espaçadas de 10 unidades temporais;
- Para cada tarefa do conjunto mínimo, é utilizado o majorante exequível de carga computacional especificado na subsecção 4.5.1, de forma a que o conjunto das tarefas seja escalonável;

- A especificação da carga computacional anterior é sucessivamente dividida por dois, quando a dimensão do conjunto de tarefas duplica, isto é, para 32 tarefas a carga computacional de cada tarefa é metade da especificada para 16 tarefas, e assim sucessivamente, de forma a manter a carga global, quer em termos relativos quer em termos absolutos;

- Por questões de simplicidade de implementação todas as tarefas devem ser criadas a partir de uma única classe, isto é, são instâncias com parâmetros temporais específicos de uma classe que define o esqueleto da tarefa sintética;

- Por último, as especificações anteriores são ainda válidas para ambos os sistemas integrados *MB-SoC* e *PPC-SoC*, ou seja, as características temporais e de carga para cada tarefa são as mesmas para os dois sistemas integrados.

4.5.2.1 Carga Relativa do Executivo OReK

A tabela 4.32 mostra a carga periódica relativa do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas e assumindo um período de 1 milisegundo.

Por comparação da tabela 4.32 com a correspondente 4.29, pode concluir-se que o aumento do período do sistema (ou diminuição da resolução temporal) num factor de dez possibilita uma grande redução da carga relativa do executivo *OReK*. No melhor caso, correspondente à implementação *OReK-PPC-DDR* com 16 tarefas, obteve-se uma redução da carga periódica relativa do executivo de, aproximadamente, 10 vezes. Relativamente aos piores casos, obteve-se uma redução da carga periódica relativa do executivo de, aproximadamente, 7 e 8 vezes, para as implementações *OReK-PPC-Cached* e *OReK-MB-Int*, ambas com 128 tarefas, respectivamente.

Implementação	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-MB-DDR*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Carga Relativa do Executivo ($T_{Tick} = 1ms$)	0,7%	1,4%	2,9%	0,8%	2,0%	0,8%	16
	1,0%	1,9%	4,0%	1,1%	3,0%	1,0%	32
	1,5%	3,0%	6,3%	1,7%	5,0%	1,5%	64
	2,6%	5,1%	10,8%	2,8%	9,0%	2,6%	128

Tabela 4.32: Carga periódica relativa do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas e assumindo um período de 1 milisegundo.

Finalmente, por comparação directa de valores presentes na tabela 4.32, constata-se que a carga relativa do executivo *OReK* na implementação *OReK-PPC-Int* é superior à análoga *OReK-MB-Int*. Esta constatação deve-se aos motivos apresentados anteriormente, na secção 4.5.1.1.

4.5.2.2 Tempo Livre Relativo

A tabela 4.33 mostra a carga periódica relativa livre nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*,

em função do número de tarefas e assumindo um período de 1 milisegundo.

Implementação	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-MB-DDR*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Tempo	44,1%	43,4%	41,9%	69,7%	68,2%	69,8%	16
Livre	43,8%	42,9%	40,8%	69,4%	67,2%	69,5%	32
Relativo	43,3%	41,8%	38,5%	68,8%	65,1%	69,0%	64
($T_{Tick} = 1ms$)	42,2%	39,7%	34,0%	67,7%	61,0%	67,9%	128

Tabela 4.33: Carga periódica relativa livre nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas e assumindo um período de 1 milisegundo.

Por comparação da tabela 4.33 com a correspondente 4.30, pode-se concluir que o aumento do período do sistema num factor de 10 possibilita um aumento considerável da carga relativa livre.

4.5.2.3 Carga Relativa das Tarefas

A tabela 4.34 mostra a carga periódica relativa das tarefas nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas e assumindo um período de 1 milisegundo.

Por comparação dos valores apresentados na tabela 4.34, constata-se que a carga relativa das tarefas nas implementações com o sistema integrado *PPC-SoC* é consideravelmente inferior às implementações com o *MB-SoC*.

A comparação da tabela 4.34 com a correspondente 4.31, mostra que os valores da carga periódica relativa das tarefas se mantiveram praticamente inalterados, satisfazendo os critérios predefinidos.

Implementação	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-MB-DDR*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Carga	55,2%	55,2%	55,2%	29,5%	29,8%	29,5%	16
Relativa das	55,2%	55,2%	55,2%	29,5%	29,8%	29,5%	32
Tarefas	55,2%	55,2%	55,2%	29,5%	29,9%	29,5%	64
($T_{Tick} = 1ms$)	55,2%	55,2%	55,2%	29,5%	30,0%	29,5%	128

Tabela 4.34: Carga periódica relativa das tarefas nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas e assumindo um período de 1 milisegundo.

4.5.3 Carga Relativa com Período de 10 *m*s

Além dos pressupostos referidos na secção 4.5 e no sentido de manter a carga periódica relativa constante, de forma análoga ao referido na secção 4.5.2, as tarefas a implementar devem ainda possuir as seguintes características temporais (ver figura 4.2):

- A configuração inicial consiste num conjunto mínimo formado por 16 tarefas, com fases iniciais compreendidas entre 0 e 15, uniformemente espaçadas de 1 unidade temporal e com um período igual à *deadline* de 16 unidades temporais (*ticks*);

- As especificações anteriores mantêm-se para implementações múltiplas do conjunto mínimo, isto é, para 32, 64 e 128 tarefas, em que são simultaneamente activadas 2, 4 ou 8 tarefas, uniformemente espaçadas de 1 unidade temporal;
- Para cada tarefa do conjunto mínimo, é utilizado o majorante exequível de carga computacional especificado na subsecção 4.5.1, de forma a que o conjunto das tarefas seja escalonável;
- A especificação da carga computacional anterior é sucessivamente dividida por dois, quando a dimensão do conjunto de tarefas duplica, isto é, para 32 tarefas a carga computacional de cada tarefa é metade da especificada para 16 tarefas, e assim sucessivamente, de forma a manter a carga global, quer em termos relativos quer em termos absolutos;
- Por questões de simplicidade de implementação todas as tarefas devem ser criadas a partir de uma única classe, isto é, são instâncias com parâmetros temporais específicos de uma classe que define o esqueleto da tarefa sintética;
- Por último, as especificações anteriores são ainda válidas para ambos os sistemas integrados *MB-SoC* e *PPC-SoC*, ou seja, as características temporais e de carga para cada tarefa são as mesmas para os dois sistemas integrados.

4.5.3.1 Carga Relativa do Executivo OReK

A tabela 4.35 mostra a carga periódica relativa do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas e assumindo um período de 10 milisegundos.

Por comparação da tabela 4.35 com a correspondente mais próxima 4.29, pode concluir-se que o aumento do período do sistema num factor de dez possibilita uma redução da carga relativa do executivo *OReK*, mas consideravelmente inferior à obtida anteriormente, na secção 4.5.2.1. No melhor caso, correspondente à implementação *OReK-PPC-DDR* com 16 tarefas, obteve-se uma redução da carga periódica relativa do executivo de, aproximadamente, 7 vezes. Relativamente aos piores casos, obteve-se uma redução da carga periódica relativa do executivo de, aproximadamente, 3 vezes, para as implementações *OReK-PPC-Cached* e *OReK-MB-Int*, ambas com 128 tarefas.

Finalmente, por comparação directa dos valores presentes na tabela 4.32, constata-se que a carga relativa do executivo *OReK* na implementação *OReK-PPC-Int* é superior à análoga *OReK-MB-Int*. Esta constatação deve-se aos motivos apresentados anteriormente, na secção 4.5.1.1.

4.5.3.2 Tempo Livre Relativo

A tabela 4.36 mostra a carga periódica relativa livre nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas e assumindo um período de 10 milisegundos.

Implementação	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-MB-DDR*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Carga	0,1%	0,2%	0,4%	0,3%	0,3%	0,2%	16
Relativa do	0,2%	0,4%	0,7%	0,6%	0,6%	0,3%	32
Executivo	0,4%	0,7%	1,3%	1,0%	1,0%	0,5%	64
($T_{Tick} = 10ms$)	0,8%	1,3%	2,4%	2,0%	2,0%	0,9%	128

Tabela 4.35: Carga periódica relativa do executivo *OReK* nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas e assumindo um período de 10 milisegundos.

Implementação	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-MB-DDR*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Tempo	44,7%	44,6%	44,4%	70,2%	70,2%	70,4%	16
Livre	44,6%	44,4%	44,1%	70,0%	70,0%	70,3%	32
Relativo	44,4%	44,1%	43,5%	69,4%	69,4%	70,1%	64
($T_{Tick} = 10ms$)	44,0%	43,5%	42,4%	68,4%	68,4%	69,6%	128

Tabela 4.36: Carga periódica relativa livre nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas e assumindo um período de 10 milisegundos.

Por comparação da tabela 4.36 com a correspondente mais próxima 4.33, pode-se concluir que o aumento do período do sistema num factor de 10 possibilita um aumento residual da carga relativa livre.

4.5.3.3 Carga Relativa das Tarefas

A tabela 4.37 mostra a carga periódica relativa das tarefas nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, em função do número de tarefas e assumindo um período de 10 milisegundos.

Por comparação dos valores apresentados na tabela 4.37, constata-se que a carga relativa das tarefas nas implementações com o sistema integrado *PPC-SoC* é consideravelmente inferior às implementações com o *MB-SoC*. A comparação da tabela 4.37 com as correspondentes 4.34 e 4.31, mostra que os valores da carga periódica relativa das tarefas se mantiveram praticamente inalterados, indo de acordo com os critérios definidos.

Implementação	*OReK-MB-Int*	*OReK-MB-OPB*	*OReK-MB-DDR*	*OReK-PPC-Int*	*OReK-PPC-DDR*	*OReK-PPC-Cached*	**Nº de Tarefas**
Carga	55,2%	55,2%	55,2%	29,5%	29,5%	29,5%	16
Relativa das	55,2%	55,2%	55,2%	29,5%	29,5%	29,5%	32
Tarefas	55,2%	55,2%	55,2%	29,6%	29,6%	29,5%	64
($T_{Tick} = 10ms$)	55,2%	55,2%	55,2%	29,7%	29,7%	29,5%	128

Tabela 4.37: Carga periódica relativa das tarefas nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached* em função do número de tarefas e assumindo um período de 10 milisegundos.

4.6 Análise dos Resultados

Dos resultados obtidos relativos à avaliação do executivo *OReK*, pode-se concluir que existem vantagens e desvantagens, associadas às diversas implementações. De uma maneira geral, as implementações *OReK-MB-Int*, *OReK-PPC-Int* e *OReK-PPC-Cached* possibilitam um menor tempo de execução em várias funções internas do executivo e serviços iniciados pela aplicação.

Por outro lado, a implementação *OReK-ARPA-C2* tipicamente apresenta menores tempos de execução, quando os parâmetros a avaliar variam com o número de tarefas. As implementações *OReK-ARPA-Sw* e *OReK-ARPA-C2* possuem uma arquitectura optimizada para sistemas de tempo-real e requerem menores requisitos de memória que as restantes implementações.

Por último, as versões *OReK-MB-DDR* e *OReK-PPC-DDR* disponibilizam maior capacidade de armazenamento em detrimento da penalização temporal adicionada, sendo particularmente úteis em sistemas com grande número de tarefas, onde uma implementação integral em memória interna pode não ser viável ou possuir um custo demasiado elevado por necessitar de *FPGAs* com maior capacidade lógica e de memória.

A figura 4.3 ilustra a carga periódica relativa do executivo *OReK* em função do número de tarefas, nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, para as resoluções temporais 0,1ms, 1ms e 10ms.

Da figura 4.3 é imediato constatar que a carga periódica relativa do executivo *OReK* cresce com o aumento do número de tarefas e diminui com o incremento do período T_{Tick}, para as diversas implementações. Além disso, para o mesmo número de tarefas e igual resolução temporal, as várias implementações ilustram diferentes valores de carga periódica relativa do executivo *OReK*.

O aumento da carga periódica relativa do executivo *OReK* em função do número de tarefas deve-se essencialmente à necessidade de percorrer maiores listas de tarefas para gestão interna do núcleo. Por outro lado, o aumento de T_{Tick} confere maior espaçamento temporal entre sucessivos instantes de execução das funções de gestão do núcleo, o que contribui para uma diminuição da carga periódica relativa do executivo *OReK*.

Por último, o facto das várias implementações apresentarem diferentes valores de carga periódica relativa do executivo *OReK*, para o mesmo número de tarefas e igual resolução temporal, deve-se às particularidades intrínsecas de cada implementação. Implementações que requerem mais tempo para acesso à memória, tais como *OReK-MB-OPB*, *OReK-MB-DDR* e *OReK-PPC-DDR*, penalizam o desempenho do processador, o que se traduz num aumento da carga periódica relativa do executivo *OReK*. Por outro lado, na implementação *OReK-MB-Int*, processador e memória operam à mesma frequência, facto que minimiza a latência no acesso ao conteúdo da memória e contribui para um aumento global do desempenho do sistema. A implementação *OReK-PPC-Int* sofre penalizações a diversos níveis, desde a memória operar a uma frequência inferior à *CPU*, até às diferenças no conjunto de instruções do processador e diferentes optimizações introduzidas pelo compilador. Como referido anteriormente, a dimensão desta implementação é aproximadamente 60% superior à análoga com o processador *MicroBlaze*. Estas penalizações são grandemente responsáveis pela diminuição do desempe-

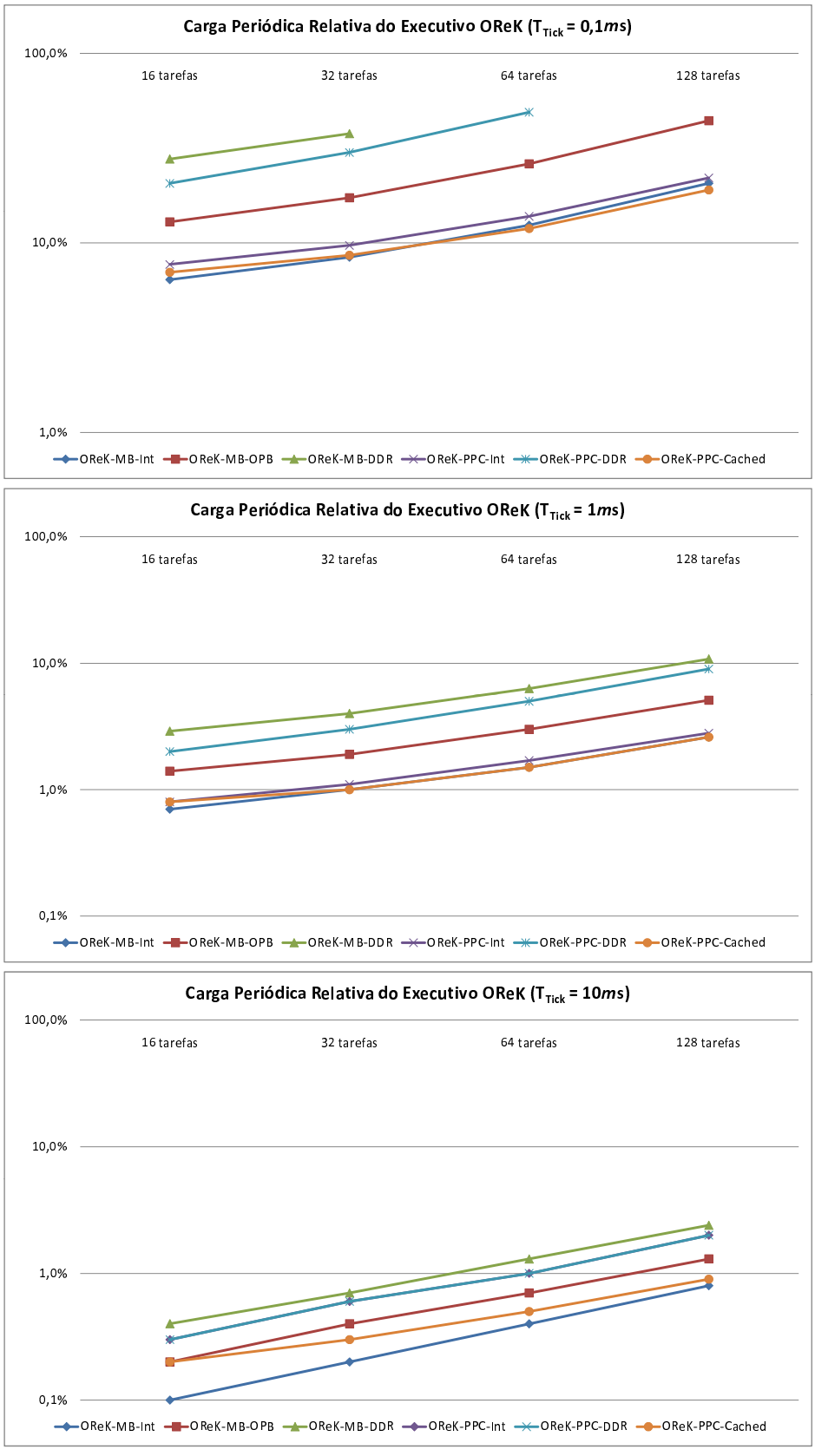


Figura 4.3: Carga periódica relativa do executivo *OReK* em função do número de tarefas, nas implementações *OReK-MB-Int*, *OReK-MB-OPB*, *OReK-MB-DDR*, *OReK-PPC-Int*, *OReK-PPC-DDR* e *OReK-PPC-Cached*, para as resoluções temporais 0,1ms, 1ms e 10ms.

nho na implementação *OReK-PPC-Int*, resultando num aumento da carga periódica relativa do executivo *OReK*. A implementação *OReK-PPC-Cached* utiliza uma memória *cache* que opera à frequência da *CPU*, conferindo-lhe um desempenho médio (não determinístico) superior à implementação *OReK-PPC-Int*, o que contribui para uma diminuição da carga periódica relativa do executivo *OReK*.

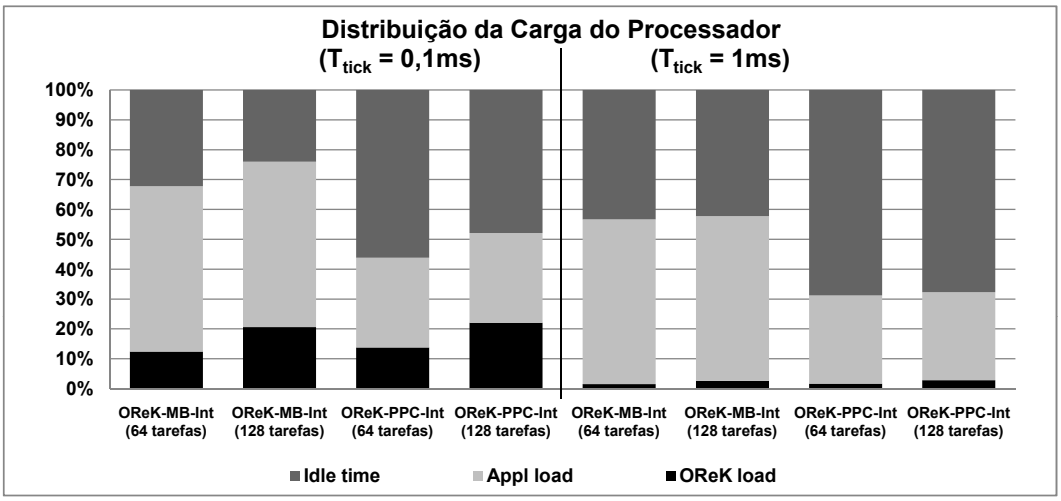


Figura 4.4: Distribuição da carga do processador nas implementações *OReK-MB-Int* e *OReK-PPC-Int*, na execução da aplicação de avaliação com 64 e 128 tarefas, para as resoluções temporais 0,1ms e 1ms.

A figura 4.4 ilustra a distribuição da carga do processador nas implementações *OReK-MB-Int* e *OReK-PPC-Int*, na execução da aplicação de avaliação com 64 e 128 tarefas, para as resoluções temporais 0,1ms e 1ms.

Observando a figura 4.4 facilmente se constata que a resolução temporal do sistema, o desempenho do processador e o número de tarefas possuem grande impacto sobre a distribuição de carga do processador. Mais concretamente, a utilização do processador *PowerPC 405* permite ganhos de desempenho significativos na execução das tarefas, ou seja, para a mesma carga computacional que as tarefas requerem em ambas as implementações, a utilização do processador *PowerPC 405* permite uma redução do seu tempo de processamento próxima de 50%. Por outro lado, observa-se que a carga do executivo é ligeiramente menor para a implementação com o processador *MicroBlaze*, que se deve grandemente à penalização do acesso à memória na implementação com o processador *PowerPC 405* e ainda aos factores apresentados na secção 4.5.1.1. Por fim observa-se que a carga do executivo *OReK* aumenta com o número de tarefas e com a resolução temporal, o que vai de acordo com o que foi anteriormente referido e justificado.

4.6.1 Comparação com Outros Executivos de Tempo-real

Dos parâmetros mais comuns na avaliação de um executivo, encontram-se a comutação de contexto e a latência até ao atendimento de uma interrupção de *hardware*. Os valores dos diversos executivos apresentados na tabela 2.1, relativos à comutação de contexto, variam entre 1μs para o *μC/OS-II* [Mic92] a 5μs para o *AVIX* [AVI06]. No executivo *OReK*, os valores

resultantes da avaliação do mesmo parâmetro variam consideravelmente com a abordagem implementada, desde 0,7μs a 11μs para as implementações *OReK-MB-Int* e *OReK-PPC-DDR*, respectivamente. Na verdade, os valores referentes à comutação de contexto são maioritariamente dependentes da arquitectura, no sentido em que o número de registos envolvidos na operação, a frequência do processador e a latência no acesso à memória condicionam o tempo de comutação de contexto. Tal facto torna evidente que, ao contrário da arquitectura utilizada, o executivo pouco contribui para este tempo.

Relativamente à máxima latência no atendimento de uma interrupção, os valores apresentados na tabela 2.1 variam entre 0μs para o *AVIX* [AVI06] a 5μs para o μ*C/OS-II* [Mic92]. No executivo *OReK*, os valores resultantes da mesma avaliação variam de acordo com a abordagem implementada, desde 0μs a 2,3μs para as implementações *OReK-ARPA-C2* e *OReK-ARPA-Sw*, respectivamente. Porém, para obter-se a máxima latência é ainda necessário considerar o maior intervalo de tempo durante o qual as interrupções podem permanecer desactivadas. No executivo *OReK* este tempo corresponde a todas as operações que são realizadas em resposta a uma interrupção gerada pelo temporizador do sistema, ou seja, salvaguarda e restauro do contexto (tabela 4.1), processamento periódico e comutação de tarefas (tabela 4.2) e selecção da próxima tarefa a executar (tabela 4.3 ou 4.4). Este varia ainda com o número de tarefas, pelo que a determinação da máxima latência depende da aplicação em concreto.

4.7 Aspectos não Avaliados

A avaliação do desempenho realizada nas secções anteriores permite determinar os majorantes de diversos parâmetros temporais do executivo *OReK*. Além disso, foi ainda realizada uma avaliação da carga periódica relativa que permite determinar, para um sistema formado pelo executivo *OReK* com uma aplicação composta por diversas tarefas concorrentes, a carga relativa do executivo, das tarefas e tempo livre restante. No entanto, esta avaliação foi baseada em tarefas sintéticas, pelo que não consiste num caso real onde existe interacção da aplicação com o mundo exterior. Além disso, o tempo de processamento de certos parâmetros do executivo *OReK* é dependente da aplicação, razão pela qual os resultados obtidos referentes à carga periódica relativa não podem ser considerados majorantes absolutos.

A variabilidade do tempo de execução de algumas funções internas e serviços nas implementações completamente realizadas em *software* possui um impacto negativo com oscilações temporais (*jitter*) na execução das tarefas do sistema e na latência no serviço de interrupções. Apesar de não ter sido avaliado, é previsível que a implementação *OReK-ARPA-C2*, sendo mais determinística, possua um *jitter* inferior na execução das tarefas da aplicação.

A determinação do consumo global de potência da aplicação e do executivo nas implementações desenvolvidas foi também um dos aspectos não cobertos na avaliação efectuada. Outro aspecto não avaliado foi a ocupação física (ou área ocupada) das diversas implementações. Estes são factores com alguma importância em sistemas integrados, na medida em que podem ser condicionantes no momento da selecção de uma implementação.

Capítulo 5

Conclusão

Sumário

Este capítulo conclui a dissertação. Começa por apresentar um resumo do trabalho realizado no âmbito deste mestrado integrado, seguidamente é feita uma discussão final dos resultados obtidos, sendo ainda mostrada a concretização dos objectivos estabelecidos no capítulo 1. Por último, são apresentadas algumas linhas de trabalho e de investigação futura.

5.1 Resumo do Trabalho Realizado

O trabalho realizado no âmbito desta dissertação pode ser resumido nos seguintes pontos:

- Desenvolvimento e implementação de uma plataforma de *hardware* que integra num único dispositivo físico o processador *MicroBlaze*, memória, periféricos e interfaces de entrada/saída;
- Desenvolvimento e implementação de uma plataforma de *hardware* que integra num único dispositivo físico o processador *PowerPC 405*, memória, periféricos e interfaces de entrada/saída;
- Adaptação e implementação do executivo de tempo-real *OReK* em ambos os sistemas integrados desenvolvidos;
- Desenvolvimento e implementação de um módulo de contadores/temporizadores especializado para medição precisa de intervalos de tempo e dotado de interfaces para possibilitar a interligação às plataformas de *hardware* desenvolvidas;
- Avaliação e estudo comparativo das várias implementações do executivo *OReK* nas diferentes plataformas.

Os dois primeiros pontos correspondem à construção dos sistemas integrados *MB-SoC* e *PPC-SoC*, respectivamente, descritos em maior detalhe no apêndice B. Para a construção destes sistemas recorreu-se a um conjunto de ferramentas de *software* denominadas por *Xilinx EDK* (*Embedded Development Kit*). Estas incluem alguns núcleos de propriedade intelectual (como o processador *MicroBlaze*, temporizadores e controladores *ethernet*) e permitem o desenvolvimento de plataformas para variados fins, com diferentes níveis de complexidade. É ainda

possível desenvolver e interligar núcleos de propriedade intelectual, o que possibilita a implementação de funções específicas ou acelerar funcionalidades já existentes.

De forma a dotar com capacidades de comunicação com o mundo exterior e de tornar os sistemas desenvolvidos realistas no sentido de poderem ser utilizados em aplicações práticas, além dos processadores *MicroBlaze* e *PowerPC 405* utilizados nos sistemas integrados *MB-SoC* e *PPC-SoC*, respectivamente, foram ainda implementados, de forma comum a ambos os sistemas, os seguintes módulos:

- Memória interna para armazenamento do código e dados do processador;
- Interface para memória externa do tipo *DDR* como alternativa para armazenamento do código e/ou dados do processador;
- Unidade de temporizadores/contadores para medir tempos e gerar sinais com frequência e/ou *duty cycle* programável;
- Controlador de interrupções;
- *UART* para implementação do protocolo *RS232*;
- Portos de entrada/saída digitais para ligação a dispositivos arbitrários;
- Módulo de depuração;
- Módulo de temporizadores especializado para medição precisa de intervalos de tempo, utilizado na avaliação do executivo.

Os sistemas integrados desenvolvidos possibilitam a execução de aplicações em *software*. Desta forma, *MB-SoC* e *PPC-SoC* podem ser utilizados como base para múltiplas implementações de sistemas embutidos, desde multimédia e telecomunicações a sistemas militares.

O executivo de tempo-real *OReK* era suportado inicialmente nas arquitecturas *Intel x86* e *ARPA*. No contexto deste trabalho, foi portado para os sistemas integrados desenvolvidos, *MB-SoC* e *PPC-SoC*. Esta operação requereu o desenvolvimento de diversas funções de baixo-nível para suporte dos processadores *MicroBlaze* e *PowerPC 405*.

Como medida da complexidade para efectuar o porte do executivo *OReK* para os dois sistemas integrados desenvolvidos, pode referir-se que consiste em cerca de 400 linhas de código em linguagem *C* e 160 linhas em linguagem *Assembly*. Além do porte foram adicionados alguns melhoramentos pontuais, como a possibilidade de desactivar o envio de informação de *log* do executivo *OReK*, com a mais valia de minimizar o tamanho do executivo (*memory footprint*).

Todas as implementações do executivo *OReK* possuem a mesma interface, o que permite que a mesma aplicação possa usar indiferentemente qualquer implementação, bastando para tal ser compilada com a biblioteca adequada. Desta forma foi possível a comparação imediata do desempenho das diversas implementações.

Com a finalidade de facilitar a medição precisa de intervalos de tempo, foi desenvolvido em *VHDL* um módulo de contadores/temporizadores, descritos no apêndice C. O módulo foi desenvolvido com as seguintes características gerais:

- Cinco contadores/temporizadores independentes;
- Dimensão de 64 bits para cada contador;
- Inicialização, activação e suspensão dos contadores, de forma individual e simultânea;
- Interface *FSL* para possibilitar comunicação de baixa latência com o processador *MicroBlaze*;
- Interface *PLB* para possibilitar comunicação com o processador *PowerPC 405*.

Todo este trabalho permitiu implementar o fluxo de co-projecto de *hardware-software*, para aplicações embutidas de tempo-real baseadas nos sistemas integrados *MB-SoC* e *PPC-SoC*, descrito no apêndice B. Uma das dificuldades na sua concretização foi o facto do trabalho ter coberto diversos níveis de abstracção, desde a modelação dos temporizadores em *VDHL*, passando pelo desenvolvimento dos sistemas embutidos com recurso a ferramentas de suporte, até ao *software* desenvolvido para as rotinas de baixo nível do executivo e para possibilitar o funcionamento e avaliação do executivo *OReK*.

5.2 Análise Final dos Resultados

O desenvolvimento dos sistemas integrados *MB-SoC* e *PPC-SoC* com suporte para um executivo de tempo-real e implementação em dispositivos lógicos programáveis resulta nas seguintes vantagens:

- Os sistemas integrados desenvolvidos são optimizados para *FPGA* e possibilitam um elevado grau de configurabilidade, quer ao nível do *hardware*, que pode ser configurado para suportar novos periféricos e dispositivos de entrada/saída ou ainda para alterar características nos módulos existentes, quer ao nível do *software* que permite implementar o executivo *OReK* em conjunto com a aplicação;
- Possuem uma elevada previsibilidade de execução motivada pela utilização da política de escalonamento *on-line* do tipo *EDF* para as tarefas de tempo-real críticas. Esta assegura o correcto funcionamento temporal do sistema no caso de este ser escalonável;
- Devido às características apresentadas nos pontos anteriores, os sistemas desenvolvidos podem ser utilizados na integração e implementação de sistemas embutidos destinados a muitos equipamentos e aplicações que exijam uma elevada fiabilidade, tais como os transportes, os sistemas de segurança, a automação industrial, ou outras.

Relativamente aos trabalhos referenciados no capítulo 2, o trabalho apresentado nesta dissertação diferencia-se nos seguintes aspectos:

- Utilização do algoritmo *EDF* para o escalonamento de tarefas de tempo-real críticas, o qual possibilita uma taxa máxima de utilização da *CPU* de 100 %, garantindo a escalonabilidade;

- Suporte para conjuntos heterogéneos de tarefas constituídos por tarefas ordinárias, tarefas de tempo-real não críticas e tempo-real críticas. Nas classes de tempo-real são ainda suportadas tarefas periódicas e aperiódicas;

- Serviço de interrupções através de tarefas aperiódicas, escalonadas em conjunto com as restantes tarefas de acordo com as restrições temporais definidas pelo programador e com restrições ao nível do tempo mínimo entre activações sucessivas da mesma fonte de interrupção;

- Elevada configurabilidade do *hardware*, proporcionada pela utilização de um dispositivo lógico reconfigurável (*FPGA*) e ainda pelas ferramentas de suporte *Xilinx EDK*.

Por fim, os resultados obtidos no capítulo 4 permitem concluir que existem vantagens e desvantagens associadas às diversas implementações, não existindo uma implementação que concilie todas as vantagens. À partida a versão *OReK-ARPA-C2* é a implementação com execução mais determinística, motivada por parte do processamento ser efectuado em *hardware* dedicado. Por outro lado, as implementações *OReK-MB-Int* e *OReK-PPC-Int* possuem diversos tempos de processamento inferiores aos da implementação *OReK-ARPA-C2*, devido à utilização de processadores de alto desempenho e optimizados para *FPGA*. Nos testes de carga periódica relativa efectuados, a carga computacional do executivo *OReK* é menor na implementação *OReK-MB-Int*. Por outro lado, o processador *PowerPC 405* tem maior capacidade computacional proporcionada pela sua arquitectura interna e integração em lógica dedicada da *FPGA*, pelo que uma implementação que execute completamente em memória *DDR* e com a *cache* activa, é expectável que possibilite, em termos médios, melhores resultados que a implementação *OReK-MB-Int*, à custa obviamente da perda de determinismo introduzido pela *cache*.

5.3 Direcções de Trabalho Futuro

Da dissertação realizada, resultaram várias ideias para possíveis linhas de trabalho e investigação futura. De seguida são propostos diversos tópicos, abrangendo vertentes quer de investigação, quer de desenvolvimento.

5.3.1 Expansão das Funcionalidades Suportadas nos Sistemas Integrados MB-SoC e PPC-SoC

Para possibilitar um leque mais abrangente de funcionalidades disponibilizadas, poderia ser adicionado suporte ao nível do *hardware* para interfaces de armazenamento estático de informação e interfaces que implementem protocolos de comunicação, tais como *CAN*, *USB* e *ethernet*. Do lado do *software* poderia ser adicionado suporte para sistemas de ficheiros e ainda ao nível de redes, como o suporte para os protocolos *TCP/IP*, *FTP*, *SNMP*, entre outros.

5.3.2 Adaptação de um Executivo de Tempo-real para os Sistemas Integrados MB-SoC e PPC-SoC

No sentido de possibilitar um estudo mais justo e completo, seria interessante a adaptação (ou porte) de uma executivo de tempo-real, tal como o *embOS* [Mic08], ou o *µC/OS-II* [Mic92], para os sistemas integrados *MB-SoC* e *PPC-SoC*. Desta forma seria possível uma avaliação e comparação directa entre os diversos executivos.

5.3.3 Projecto de um Coprocessador de Gestão do RTOS Baseado Numa APU de um Processador PowerPC

Os processadores *PowerPC* disponibilizam uma forma de estender a sua capacidade de processamento e conjunto de instruções (*instruction set*), através da implementação de unidades funcionais designadas por *APUs*. Esta facilidade pode ser explorada para combinar um processador de uso geral com um coprocessador especializado.

A utilização de dispositivos de *hardware* especializados para operações de gestão em sistemas de tempo-real, como o *ARPA-OSC* [Arn07] ou o *Sierra RTOS* [Pre07], implementados através de uma *Auxiliary Processing Unit* (*APU*) de um processador *PowerPC* embutido em *FPGA*, é expectável que resulte em ganhos interessantes de desempenho.

5.3.4 Conclusão da Avaliação

A conclusão da avaliação deverá cobrir os aspectos não avaliados no capítulo 4, principalmente, ao nível da eficiência energética e *jitter* das actividades periódicas do sistema. Por último, a utilização de *benchmarks* para sistemas embutidos, tais como a *EEMBC* [EEM08] ou a *MiBench* [MRG01] podem complementar esta avaliação.

Nos apêndices que se sequem é fornecida diversa informação adicional. É apresentado o sistema integrado *ARPA-SoC*, utilizado em duas das plataformas cuja avaliação é reportada no capítulo 4. São descritos os sistemas integrados desenvolvidos no âmbito desta dissertação, o *MB-SoC* e o *PPC-SoC*. Por último, é exposto o módulo de temporizadores desenvolvido para medir de forma precisa intervalos de tempo.

Apêndice A

O Sistema Integrado ARPA-SoC

A.1 Introdução

O sistema integrado *ARPA-SoC* [MIP04] (*Advanced Real-time Processor Architecture - SoC*), sendo destinado a sistemas embutidos, integra numa única *FPGA* o processador *ARPA-MT* (implementação *Simultaneous Multithreading pipelined* da arquitectura *MIPS32* com os coprocessadores 0 e 2) e um conjunto de periféricos que permitem a ligação a dispositivos externos para efeitos de comunicação com outros sistemas computacionais, sensores, actuadores e outros.

É ainda importante salientar que a plataforma *ARPA* não foi empregue no contexto deste trabalho, estando a sua descrição enquadrada nesta dissertação como acréscimo de informação e de valor, uma vez que existem dados divulgados da avaliação do desempenho do executivo *OReK* nesta plataforma, o que possibilita uma comparação directa com o trabalho efectuado. O sistema integrado *ARPA-SoC* foi concebido e implementado no âmbito de uma tese de doutoramento [Arn07], desenvolvida na Universidade de Aveiro.

O seu desenvolvimento foi feito em *VHDL* através da elaboração de modelos ao nível *RTL* (*Register Transfer Level*) para cada um dos seus módulos constituintes [Arn07]. Estes modelos são sintetizáveis e em grande parte independentes da tecnologia de implementação alvo. A sua prototipagem foi realizada numa *FPGA XC3S1500-4BG676* da família *Spartan-3* da *Xilinx* [Xil08l].

A.2 Estrutura Interna

O projecto *ARPA-SoC* está implementado de forma hierárquica. Neste apêndice será considerado apenas o nível hierárquico superior. Os restantes níveis hierárquicos do processador e ainda a estrutura interna e implementação dos periféricos não é relevante para esta discussão.

A figura A.1 ilustra a estrutura interna do nível hierárquico superior do sistema e estão interligados os seguintes módulos:

- Processador - inclui os coprocessadores *Cop0-MEC* e *Cop2-OSC* e a *CPU ARPA-MT*;

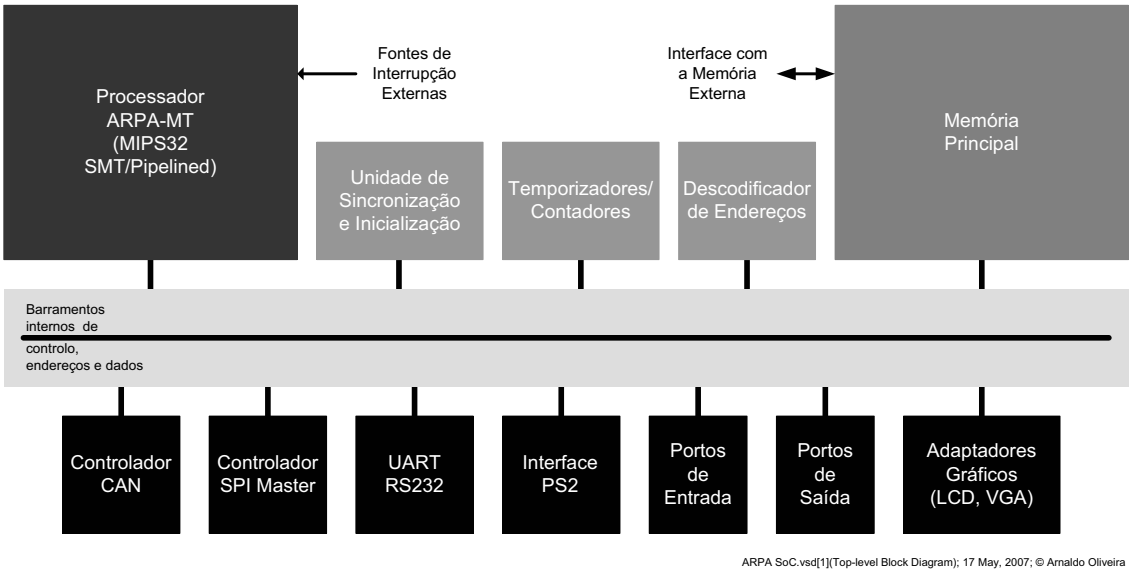


Figura A.1: Diagrama de blocos do nível hierárquico superior do sistema integrado *ARPA-SoC*, (retirado de [Arn07]).

- Unidade de Sincronização e Inicialização - gera os sinais de inicialização e sincronização do processador de forma flexível e controlada por programação;
- Unidade de temporizadores/contadores - usados para contar eventos, medir tempos e gerar sinais com frequência e/ou *duty cycle* controlado por *software*;
- Descodificador de endereços - gera as linhas de selecção da memória e dos periféricos a partir dos sinais do barramento de endereços;
- Memória Principal - utilizada para armazenar código e dados da aplicação;
- Controlador *CAN* - implementa o protocolo *CAN* (*Controller Area Network*) 2.0 - Parte B [Rob08] usado em barramentos de campo para sistemas de controlo distribuído;
- Controlador *SPI Master* - implementa as funcionalidades de um *master SPI* (*Serial Peripheral Interface*) para ligação a periféricos externos (e.g. *ADCs*, *DACs*, etc.);
- *UART RS232* - implementa o protocolo RS232 para comunicação série de baixa velocidade em modo *full-duplex*;
- Interface *PS2* - implementa a norma *PS2* para ligação de teclados e ratos;
- Portos de entrada/saída - disponibilizam um conjunto de entradas e saídas digitais para ligação a dispositivos arbitrários;
- Adaptador de *LCD* - permite a ligação a painéis *LCD* (*Liquid Crystal Display*);
- Adaptador *VGA* - permite a ligação a monitores *VGA* (*Video Graphics Adapter*).

O módulo "Memória Principal" implementa as memórias de código e de dados do processador com base nos blocos de memória internos da *FPGA*. Além disso, disponibiliza também um conjunto de portos para ligação a memórias externas de forma a expandir a memória disponível para as aplicações a executar no processador *ARPA-MT*. Na implementação actual, a memória interna da *FPGA* utilizada como memória principal do sistema é apenas de 16 KBytes (8 blocos de 16 Kbits), sendo os restantes blocos de memória da *FPGA* utilizados pelos periféricos e pelo coprocessador *Cop2-OSC*.

O módulo do processador é constituído internamente pela *CPU* e pelos coprocessadores *Cop0-MEC* e *Cop2-OSC*.

A *CPU ARPA-MT* possui uma arquitectura de 32 bits, podendo endereçar um máximo de 4 GBytes de memória externa. De referir ainda que todos os periféricos estão mapeados no espaço de endereçamento de memória, definido pelo descodificador de endereços.

A.3 Fluxo de Projecto

A utilização da plataforma *ARPA-SoC* é suportada por um fluxo de projecto representado na figura A.2 para aplicações embutidas baseadas no sistema integrado *ARPA-SoC*. De notar que a figura não apresenta as etapas de desenvolvimento do modelo do sistema integrado *ARPA-SoC* nem do respectivo *software* de suporte, centrando-se apenas na implementação das aplicações.

As metades esquerda e direita da figura A.2 ilustram os subfluxos de *hardware* e de *software*, respectivamente. A plataforma de implementação/prototipagem é mostrada na parte inferior da figura A.2, consistindo numa placa "Xilinx Spartan-3 Development Kit" comercializada pela *Avnet* [Avn05]. Além da *FPGA Spartan-3* XC3S1500 da *Xilinx*, a placa possui uma memória *ISP* (*In System Programmable)* para armazenamento não volátil de configurações da *FPGA*, memórias *SRAM* e *FLASH* e um conjunto de componentes auxiliares, tais como reguladores, osciladores, controladores, *drivers* de linha e fichas para ligação a dispositivos externos.

A.3.1 O Subfluxo de Hardware

O primeiro passo no projecto do *hardware* do sistema é a especialização dos parâmetros da plataforma do processador *ARPA-MT*, assim como a selecção dos periféricos pretendidos.

Seguidamente, são realizadas a síntese e a implementação do modelo do *hardware* com as ferramentas integradas no ambiente *ISE* (*Integrated Synthesis Environment*) da *Xilinx*. O resultado desta fase do projecto é um ficheiro com extensão ".bit" contendo a sequência de bits de configuração da *FPGA* (*ARPASoC.bit*) e um ficheiro com informação sobre a implementação e o posicionamento dos blocos de memória interna e respectivas gamas endereços (*ARPASoC_bd.bmm*).

Para possibilitar a programação da *FPGA* e funcionamento consequente do processador é necessário preencher os blocos de memória internos da *FPGA* a partir do endereço de arranque com código e dados válidos. Esta tarefa é realizada com a aplicação *data2mem* da *Xilinx*, a qual permite preencher as secções relativas aos blocos de memória dentro de um

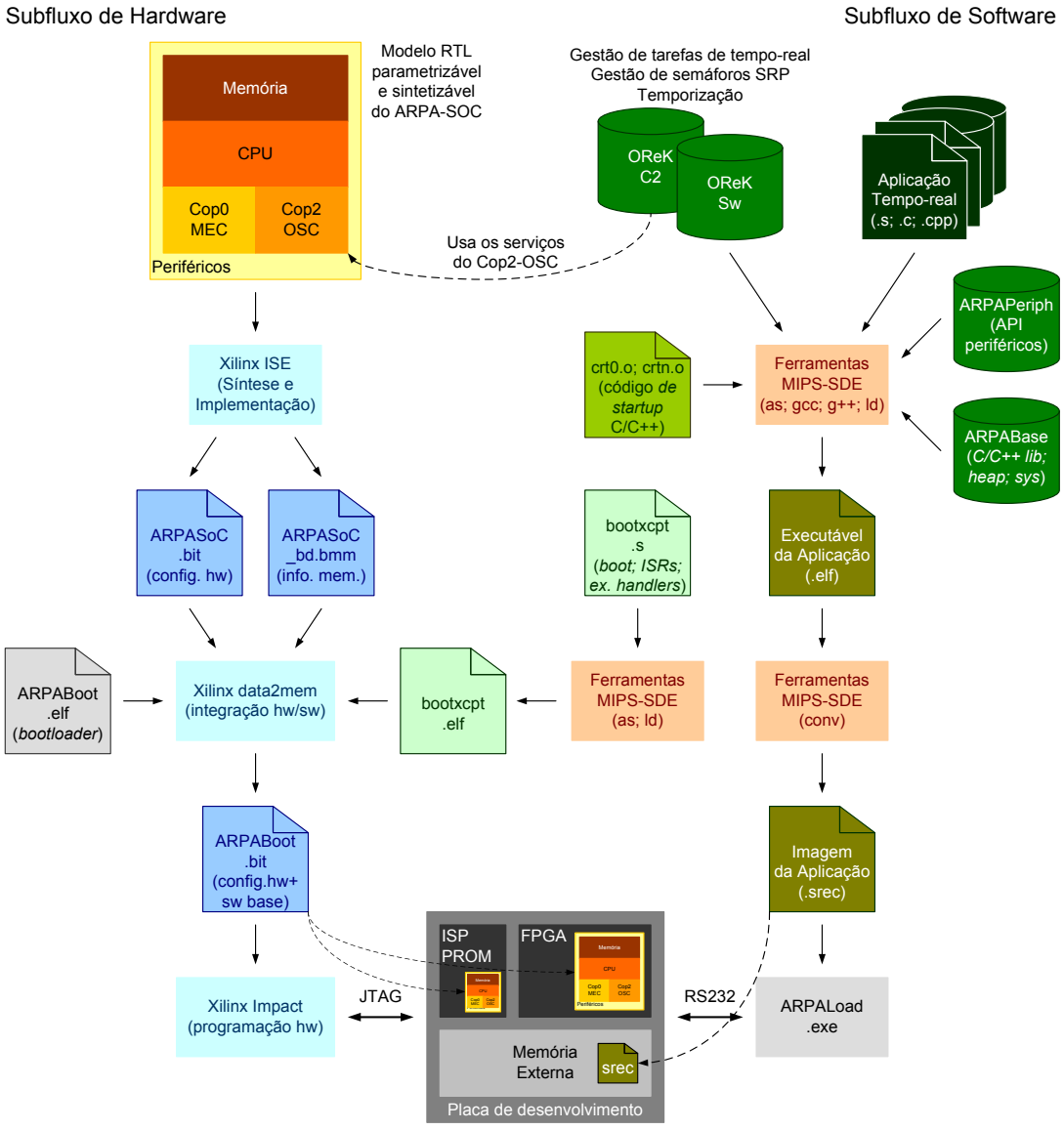


Figura A.2: Fluxo de projecto para aplicações baseadas no sistema integrado *ARPA-SoC*, (retirado de [Arn07]).

ficheiro ".bit" com a informação contida num outro ficheiro com a extensão ".elf" (*Executable and Linkable Format*), especificado na linha de comando. O ficheiro com a extensão ".elf" corresponde a módulos executáveis gerados pelas ferramentas de compilação da arquitectura *MIPS*.

Após o preenchimento dos bits correspondentes aos blocos de memória com código e dados no ficheiro *ARPASoC.bit* obtém-se o ficheiro *ARPABoot.bit* este integra a configuração do *hardware*, o código e os dados do *software* base do sistema. Por fim, este ficheiro pode ser utilizado pela aplicação *Impact* da *Xilinx* para configurar, através de uma interface *JTAG* e de um cabo adequado, a *FPGA* ou a sua memória *ISP* de configuração.

A.3.2 O Subfluxo de Software

O subfluxo de *software* é quase todo baseado na cadeia de ferramentas *MIPS-SDE*, as quais executam num computador pessoal em *linux* ou *Cygwin* (`www.cygwin.com`), permitindo:

- Compilar o código fonte de cada um dos módulos que constituem a aplicação, escritos em *assembly*, *C* ou *C++* para código objecto da arquitectura *MIPS*;
- Fazer a integração dos vários módulos objecto da aplicação, eventualmente com bibliotecas (ficheiros "*.a"), num único módulo executável (ficheiro "*.elf");
- Realizar a conversão do módulo executável para o formato *Motorola S-Record* (ficheiro "*.srec') de forma a poder ser transferido para a memória do sistema alvo com o auxílio da aplicação *ARPALoad.exe*, desenvolvida especificamente para comunicação via RS232 com o *bootloader* residente na placa de desenvolvimento.

As ferramentas de compilação *MIPS-SDE* são baseadas nas aplicações da cadeia de compilação da *GNU*. A *MIPS Technologies* mantém a sua própria base de código fonte, sincronizando-a periodicamente com as actualizações da *GNU* e concentrando-se nos aspectos específicos das ferramentas de forma a optimizá-las para a arquitectura *MIPS32*. De referir ainda que as ferramentas *MIPS-SDE* aproximam-se do *standard POSIX* [MIP04].

Estão também disponíveis diversas bibliotecas com as seguintes funcionalidades:

- Conjunto reduzido de funções da biblioteca da linguagem *C*, incluindo informação temporal e manipulação de *strings*;
- Diagnóstico, configuração do processador, controlo de interrupções e geração de excepções;
- Gestão de memória dinâmica usando as funções `malloc()` e `free()` da linguagem *C* ou os operadores `new` e `delete` da linguagem *C++*;
- Configuração e transferência de dados entre a *CPU* e os periféricos.

Estas bibliotecas são integradas juntamente com os módulos objecto da aplicação pelo *linker* sendo construído o módulo executável da aplicação.

Apêndice B

Os Sistemas Integrados MB-SoC e PPC-SoC

Sumário

Este apêndice apresenta as ferramentas de *software Xilinx EDK* (*Embedded Development Kit*) e resume a arquitectura dos processadores *MicroBlaze* [Xil08e] e *PowerPC 405* [Xil08i]. São expostas as principais aplicações disponibilizadas pela *Xilinx EDK*, nomeadamente a aplicação *XPS* (*Xilinx Platform Studio*) que inclui diversas ferramentas de *software* e núcleos de propriedade intelectual (*IP Cores*) necessários para a construção de uma plataforma de *hardware* e a aplicação *SDK* (*Software Development Kit*), que disponibiliza uma interface adequada para o desenvolvimento de projectos de *software*. Finalmente, é apresentado o fluxo de projecto e expostos os sistemas embutidos desenvolvidos, o *MB-SoC* e o *PPC-SoC*.

B.1 Introdução

Como acima referido, a *Xilinx* disponibiliza a ferramenta de suporte *EDK* como base para a construção de uma plataforma que pode incluir diversos núcleos de propriedade intelectual. Desta forma é possível construir plataformas para variados fins e diferentes níveis de complexidade, desde arquitecturas simples com um só processador e com um número reduzido de núcleos de propriedade intelectual para, por exemplo, monitorização de um processo industrial ou controlo básico de um robot, a sistemas complexos com vários processadores, interligados por uma ou mais *Network-on-Chip - NoC*. Estes possuem diferentes funcionalidades e um número superior de núcleos de propriedade intelectual, com possíveis implementações, por exemplo, em veículos automóveis onde são implementadas funções críticas que passam desde o controlo do motor, controlo de estabilidade do veículo e controlo de activação dos *airbags*, até ao controlo de funções não críticas como a monitorização da temperatura para actuar no sistema de ar-condicionado ou sistemas de comunicação sem fios que possibilitam a comunicação entre o veículo e o mundo exterior.

É ainda possível desenvolver e interligar núcleos de propriedade intelectual especializados, possibilitando o desenvolvimento de funções específicas ou acelerar funcionalidades já existentes. Esta flexibilidade resulta em plataformas com configurações virtualmente ilimitadas, sendo a capacidade intrínseca do dispositivo físico (a *FPGA*) a principal limitação.

B.2 Ferramentas de Suporte

De modo a simplificar a construção e o desenvolvimento de uma plataforma, a *Xilinx* disponibiliza duas ferramentas centrais: *Xilinx Platform Studio* e *Xilinx Software Development Kit*. A primeira implementa a plataforma de *hardware*, isto é, é responsável pela criação de um sistema base de *hardware* que inclui a configuração do processador, memória, dispositivos de entrada/saída e periféricos. Esta pode ainda ser utilizada para desenvolvimento de projectos de *software*, porém a sua interface gráfica é mais limitativa, pelo que em projectos mais complexos, é recomendada a utilização da ferramenta *SDK*. A segunda é dedicada ao desenvolvimento de projectos em *software*. A ferramenta *SDK* suporta projectos desenvolvidos em linguagem *"C++"*, *"C"* e *"Assembly"*, os quais são compilados em conjunto com as bibliotecas e *drivers* da plataforma de *hardware* previamente desenvolvida.

Após a conclusão das plataformas de *hardware* e de *software*, é possível gerar um ficheiro do tipo *bitstream* que inclui a plataforma de *hardware* para configuração da *FPGA* e a aplicação a executar na plataforma. A figura B.1 ilustra o fluxo básico no processo de um projecto embutido.

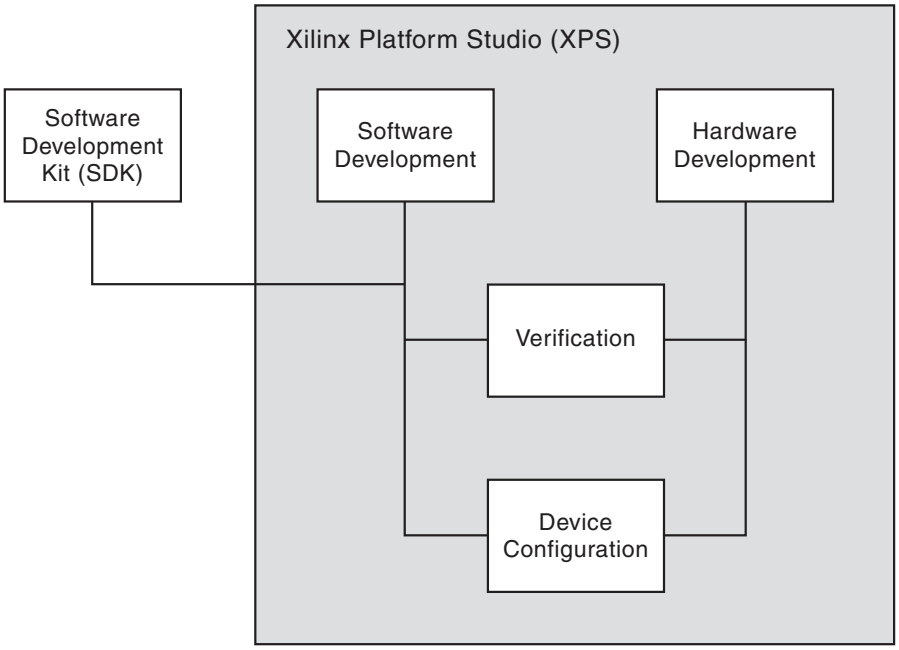


Figura B.1: Fluxo básico de projecto baseado no *EDK* (retirado de [Xil08h]).

B.2.1 A Ferramenta XPS

A ferramenta *XPS* possibilita a construção de uma plataforma de *hardware* e ainda a sua integração com a aplicação de *software*, permitindo o desenvolvimento de um sistema integrado para uma aplicação concreta.

Apesar da ferramenta *XPS* permitir a gestão de projectos de *software*, é recomendado o uso da ferramenta *SDK*. Esta foi desenvolvida especificamente para a plataforma de *software*,

sendo bastante mais completa que a anterior. Por esse motivo, a camada de *software* e assuntos relacionados só serão abordados na ferramenta *SDK*.

É ainda importante referir que as próximas subsecções resumem os diversos passos necessários para a construção de um projecto baseado no *software* de desenvolvimento *Xilinx EDK*. Assim sendo, apenas pretendeu-se dar uma visão geral que possibilite o enquadramento com o trabalho efectuado. Por outro lado, a *Xilinx* disponibiliza documentação [Xil08b] com uma extensa descrição dos conceitos, ferramentas e técnicas para utilização do *EDK*.

B.2.1.1 Criação de um Projecto de Hardware

No sentido de facultar a criação de plataformas dedicadas para vários fins, a ferramenta *XPS* disponibiliza diversos mecanismos complementares para a criação e o desenvolvimento de projectos de *hardware*.

Actualmente é possível criar plataformas de *hardware* de forma quase automática para placas de desenvolvimento com suporte integral pela ferramenta, isto é, a ferramenta contém informação integral da placa de desenvolvimento (memórias, periféricos, pinos utilizados da *FPGA* e outros), sendo apenas necessário seguir alguns assistentes (*wizards*) para implementar as funcionalidades de *hardware* desejadas na plataforma a desenvolver.

Para as placas de desenvolvimento em que a ferramenta *XPS* apenas suporta a *FPGA*, existe a possibilidade de criar um projecto vazio, onde o desenvolvimento da plataforma é inteiramente manual. Neste caso é necessário adicionar diversos núcleos de propriedade intelectual para formar o sistema pretendido. Existe ainda um método intermédio para a criação de um sistema base. Este é aplicável nos casos em que a ferramenta contém *wizards* de suporte à *FPGA*, mas não inclui informação do conteúdo do dispositivo físico. Este é o caso típico para as placas desenvolvidas por outros fabricantes, como por exemplo a *Celoxica RC10* [Cel08] ou a *XESS XSA-3S1000* [XES08]. Este método permite criar um projecto que implementa todas as funcionalidades que o utilizador especificar.

Posteriormente, para os casos em que a ferramenta não possui informação integral da placa de desenvolvimento, é necessário adicionar a informação das restrições físicas: Os pinos de ligação da *FPGA* aos componentes da placa e o número de posição da *FPGA* na cadeia *JTAG*.

Após a invocação da ferramenta *XPS* é necessário escolher o tipo de projecto a desenvolver. Começando pela opção recomendada, *Base System Builder - BSB* e admitindo que a ferramenta suporta *wizards* para a *FPGA* em questão, a criação de um projecto passará então pelo seguimento de *wizards* gráficos.

1. Selecção do Processador - Após a descrição da *FPGA*, o primeiro passo é a escolha de um processador, o qual deve ser feita de acordo com as características pretendidas para a plataforma. Os processadores disponíveis são o *MicroBlaze* e eventualmente, o *PowerPC*. Este último é um processador implementado em lógica dedicada e integrado em apenas algumas *FPGAs*, tais como as da família *Virtex-II Pro* e *Virtex-4 FX*, razão pela qual nem sempre é possível a selecção deste processador.

O *MicroBlaze* é um processador *soft-core*, o mesmo será dizer que o seu núcleo é sintetizável e que pode ser implementado nos recursos programáveis das *FPGAs*. Ambos os processadores irão ser posteriormente abordados em maior detalhe.

2. Configuração do Processador - Depois da escolha do processador, é necessário especificar parâmetros como a frequência de referência existente no dispositivo físico, a frequência de funcionamento desejada para o processador e a polaridade do *reset* que irá actuar no processador.

Por fim, é possível escolher um método de depuração, definir a quantidade de memória interna à *FPGA* com ligação dedicada ao processador e activar *cache* e uma unidade de vírgula flutuante (se desejável). Relativamente à depuração, esta pode ser feita utilizando um módulo de *hardware* dedicado ou por por intermédio de um *stub* entre o processador e a aplicação. Devido ao facto da depuração estar intimamente ligada com a camada de *software*, esta será abordada em maior detalhe na ferramenta *SDK*.

3. Selecção de Interfaces e Periféricos Externos à FPGA - Segue-se a escolha de interfaces de entrada/saída. Neste segmento é possível adicionar uma panóplia de dispositivos: Desde controladores de memória externa, rede ethernet, módulos genéricos de entrada/saída (*GPIO*), barramentos *IIC* e *SPI*, ligações e árbitros *PCI*, dispositivos para cartões de memória *Compact Flash* e comunicação *UART* via *RS232*. Depois de adicionado, cada dispositivo é ainda configurável no sentido de ir ao encontro das características desejadas para o projecto a desenvolver.

4. Selecção de Periféricos Internos à FPGA - Nesta secção encontram-se os seguintes periféricos: Controladores de memória *BRAM* e temporizadores com funções do tipo contador e *watchdog*. Os periféricos são igualmente configuráveis, de forma a irem ao encontro das características desejadas para o projecto a desenvolver.

Refira-se ainda que está implícita a presença de um controlador de interrupções, este é automaticamente adicionado se algum dispositivo for configurado para suportar interrupções.

5. Configuração do Software - Finalmente, segue-se a configuração de quais os dispositivos a utilizar como entrada e saída padrão (*stdin* e *stdout*) e a escolha da memória que irá conter o código de arranque do processador. Os dispositivos que podem ser utilizados como entrada e saída padrão são a interface de comunicação *RS232* e o módulo de depuração que permite emular uma interface *RS232*.

É ainda possível incluir aplicações de teste automático para as memórias e periféricos adicionados.

B.2.1.2 Núcleos de Propriedade Intelectual

De uma maneira geral e para projectos com algum grau de especificidade, o projecto baseado em assistentes dificilmente contém todos os dispositivos necessários à sua implementação. Assim sendo, torna-se necessário adicionar manualmente os núcleos de propriedade intelectual (*IP Cores*) que contêm os dispositivos a implementar.

No catálogo da ferramenta *XPS* estão disponíveis os seguintes núcleos de propriedade intelectual:

- Processamento - processadores *MicroBlaze* e *PowerPC 405*;
- Analógicos - conversores *ADC* e *DAC*;

- Barramentos e interligações - barramentos *FSL*, *OPB* e *PLB*, com interligações entre os diversos tipos de barramentos;
- Relógio - gerador de relógio, gestor de relógio digital (*DCM*);
- *Reset* - módulo de *reset* para o processador *PowerPC 405*;
- Interrupções - controlador de interrupções;
- Comunicação de alta velocidade - desde controladores *ethernet 10/100/1000*, controladores *CAN*, e dispositivo *USB 2.0*;
- Comunicação de baixa velocidade - interfaces *IIC*, *SPI* e *UART*;
- *Debug* - *Agilent Trace Core*, diversos núcleos de depuração da *Chipscope* e módulo de depuração do processador *MicroBlaze* (*MDM*);
- Acesso directo à memória - controladores de acesso directo à memória (*DMA*);
- Temporizadores - temporizadores de tempo fixo (*FIT*), contadores e *watchdog*;
- Entrada e saída - módulos genéricos de entrada e saída;
- Comunicação entre processos e processadores - módulos *Mailbox* e *Mutex*;
- Memória e controladores de memória - módulo de memória *BRAM* e controladores para diversos tipos de memória;
- *PCI* - árbitros *PCI* e módulo de interligação entre *PCI* e *PLB*;
- Controladores de periféricos - módulos de controladores para periféricos externos;
- Diversos - Utilitários de lógica personalizável e de separação de barramentos.

Para além dos núcleos de propriedade intelectual disponibilizados pela ferramenta *XPS*, é ainda possível adquirir e desenvolver novos núcleos.

B.2.1.3 Verificação da Plataforma de Hardware

Após a conclusão do projecto de *hardware*, é possível visualizar a plataforma construída. Esta aparece sob a forma de um diagrama de blocos e permite constatar, de uma forma simples e rápida, todos os dispositivos implementados no projecto (ver exemplo ilustrado na figura B.5).

A plataforma de *hardware* desenvolvida pode ser verificada por intermédio de ferramentas de simulação e de análise:

- Simulação - a ferramenta *XPS* possibilita a criação de um modelo de simulação que pode ser analisado posteriormente, a partir de simuladores de *HDL* como a ferramenta *ModelSim*. O modelo de simulação pode ser do tipo comportamental, estrutural ou temporal. Na simulação do sistema, o processador executa o código da aplicação que o utilizador desenvolveu;

- Análise - a ferramenta *XPS* possui diversos núcleos de propriedade intelectual dedicados à depuração. Desta forma, é possível analisar módulos implementados no dispositivo físico. Este é um método de verificação mais rigoroso, uma vez que que retira informação do sistema implementado no dispositivo físico, porém é mais limitativo, dado que ocupa recursos físicos e apenas fornece informação dos dispositivos discriminados.

B.2.1.4 Configuração do Dispositivo Físico

Depois de sintetizada a plataforma de *hardware*, é gerado um ficheiro do tipo *bitstream* que inclui todas as informações da plataforma de *hardware*, necessárias para a implementação no dispositivo físico.

B.2.2 A Ferramenta SDK

O *SDK* é um ambiente de desenvolvimento integrado e complementar ao *XPS*. É dedicado ao desenvolvimento de projectos de *software* e dotado de uma interface gráfica com diversas funcionalidades que facilitam a criação de projectos de *software*. Após a invocação desta ferramenta, são automaticamente geradas as bibliotecas e os *drivers* referentes à plataforma de *hardware* previamente desenvolvida, com a vantagem de estes já estarem disponíveis no momento da compilação do projecto.

B.2.2.1 Criação do Projecto de Software

A ferramenta *SDK* possui *wizards* para a criação de novos projectos de *software*. O primeiro passo é a escolha do tipo de projecto, este pode ser para aplicações escritas em *C* ou *C++*. Seguidamente é discriminado o processador a utilizar (no caso de uma plataforma com múltiplos processadores) e especificado o nome e localização do projecto. Em seguida, seleccionam-se as configurações desejáveis para o desenvolvimento do projecto:

- *Debug* - útil para a fase de desenvolvimento do projecto, o principal interesse reside em detectar e corrigir erros da aplicação;
- *Release* - importante para a fase final, onde é necessário minimizar o tamanho e optimizar o tempo de execução da aplicação;
- *Profile* - útil para avaliação da aplicação.

Após a escolha das configurações, fica concluída a etapa de criação do projecto. A ferramenta *SDK* cria automaticamente uma estrutura de directórios e o projecto inclui um ficheiro de demonstração com o nome *"main.c"*. De forma igualmente automática, o projecto é compilado e consequentemente, gera um ficheiro do tipo *Executable and Linkable Format*.

B.2.2.2 Definições da Plataforma de Software

Neste módulo é possível configurar definições da plataforma, do sistema operativo e dos *drivers* a utilizar na plataforma.

As configurações da plataforma de *software* são diversas, desde a escolha do *driver* do processador até à definição do compilador a utilizar e da frequência interna do processador. Existe ainda a possibilidade de escolher um dos seguintes núcleos ou sistemas operativos que servirá de suporte ao projecto de *software*:

- *Standalone* - fornece funções básicas relacionadas com o processador e funções básicas do sistema operativo, como a entrada e saída padrão. É útil para projectos simples, ou para projectos que incluam o sistema operativo;
- *Xilkernel* - fornece, além das funções básicas do ponto anterior, serviços idênticos ao standard *POSIX* tais como gestão de tarefas, escalonamento e sincronização [Xil08h]. É útil para aplicações que requeiram funcionalidades de um *kernel*.

Nas configurações do sistema operativo é possível especificar, desde os dispositivos de entrada e de saída padrão, a especificidades do sistema operativo seleccionado.

A configuração dos *drivers* permite especificar, de forma individual, o *driver* para o respectivo dispositivo físico.

B.2.2.3 O Linker Script

O *Linker Script* é um utensílio indispensável ao desenvolvimento de um projecto de *software*. Este permite atribuir segmentos da aplicação a diferentes memórias, isto é, permite definir para cada segmento da aplicação (*text*, *data*, *heap* e *stack*), qual a memória a utilizar. Desta forma, é possível optimizar o sistema a um nível global: Pode-se colocar todos os segmentos da aplicação na memória interna do processador, ou então, no caso da memória não ter capacidade suficiente, racionalizar a sua utilização. Esta solução passa por colocar na memória mais rápida, apenas os segmentos que a aplicação acede mais frequentemente, como é o caso do segmento de código. O *Linker Script* permite ainda definir o tamanho a reservar para os segmentos de memória *heap* e *stack*.

B.2.2.4 Depuração do Projecto de Software

A ferramenta *SDK* possui um utilitário dedicado para a depuração de microprocessadores (*XMD*) e que utiliza *wizards* para a criação de configurações de depuração. Na criação de uma nova configuração é necessário especificar qual o projecto/aplicação para fazer a depuração e o tipo de conexão a utilizar. Para o processador *MicroBlaze* são permitidas as seguintes conexões:

- *MicroBlaze Hardware* - a conexão é efectuada através do módulo de depuração implementado na plataforma de *hardware* (*MDM*);
- *MicroBlaze ROM Monitor* - a conexão é efectuada por intermédio de um *stub*, isto é, através de uma aplicação de *software* (*xmdstub*);
- *MicroBlaze Simulator* - a conexão é efectuada através de um simulador do processador *MicroBlaze*.

B.2.2.5 Configuração do Dispositivo Físico

Depois de compilado o projecto de *software*, é gerado um ficheiro do tipo *Executable and Linkable Format* que inclui as informações de código e dados da plataforma de *software*, necessárias para o funcionamento do sistema implementado no dispositivo físico.

Após a conclusão das plataformas de *hardware* e de *software*, é possível gerar um novo ficheiro do tipo *bitstream* para configuração do dispositivo físico. Isto é, o ficheiro gerado

resulta de uma actualização do ficheiro do tipo *bitstream*, correspondente à junção do ficheiro resultante da síntese do projecto de *hardware*, com o ficheiro do tipo *Executable and Linkable Format*, resultante da compilação do projecto de *software*. Este novo ficheiro contém todas as informações necessárias para implementar o sistema projectado no dispositivo físico.

Para efeitos de prototipagem basta descarregar o ficheiro de configuração no dispositivo físico. Numa implementação final é possível o armazenamento do ficheiro de configuração numa memória não volátil, ligada à *FPGA*, para a configuração da mesma no arranque do sistema.

B.3 O Processador MicroBlaze

O *MicroBlaze* é um processador *RISC* de *32* bits, dotado de uma arquitectura de memória do tipo *Harvard* e dispõe de um conjunto de instruções optimizadas para sistemas embutidos. O processador é disponibilizado na forma de um núcleo sintetizável, para implementações em *FPGAs*.

A versão inicial data de 2002 [Xil08e] e desde então tem sofrido sucessivos melhoramentos [Xil08e]. Actualmente encontra-se disponível a versão `7.10.a`. As principais funcionalidades suportadas são:

- Unidade de gestão de memória (*MMU/MPU*) - possibilita a execução do *software* de forma mais segura e robusta.
- Aceleração de *hardware* via *FSL* - permite ligações de baixa latência a coprocessadores dedicados para optimizar o desempenho de funções críticas.
- Unidade de vírgula flutuante (*FPU*) - compatível com o standard `IEEE-754` de precisão simples [Xil08e], optimizada para aplicações embutidas. Permite o desenvolvimento de aplicações que operam com números em vírgula flutuante.
- Configurabilidade do *hardware* utilizado pelo processador - possibilita especificar quais os módulos de *hardware* a incluir no processador. Desta forma é possível obter uma plataforma de *hardware* optimizada e que utiliza apenas os recursos de *hardware* necessários.

B.3.1 Arquitectura do Processador

A figura B.2 ilustra a arquitectura interna da versão actual do processador *MicroBlaze*.

B.3.1.1 Tipos de Dados e sua Formatação

O processador *MicroBlaze* utiliza o formato *Big-Endian*, para representação de dados. Suporta os seguintes tipos de dados:

- `Word` - com 32 bits de dimensão, o índice 0 corresponde ao bit mais significativo (*MSb*) e 31 ao bit menos significativo (*LSb*);
- `Halfword` - com 16 bits de dimensão, o índice 0 corresponde ao *MSb* e 15 ao *LSb*;
- `Byte` - com 8 bits de dimensão, o índice 0 corresponde ao *MSb* e 7 ao *LSb*.

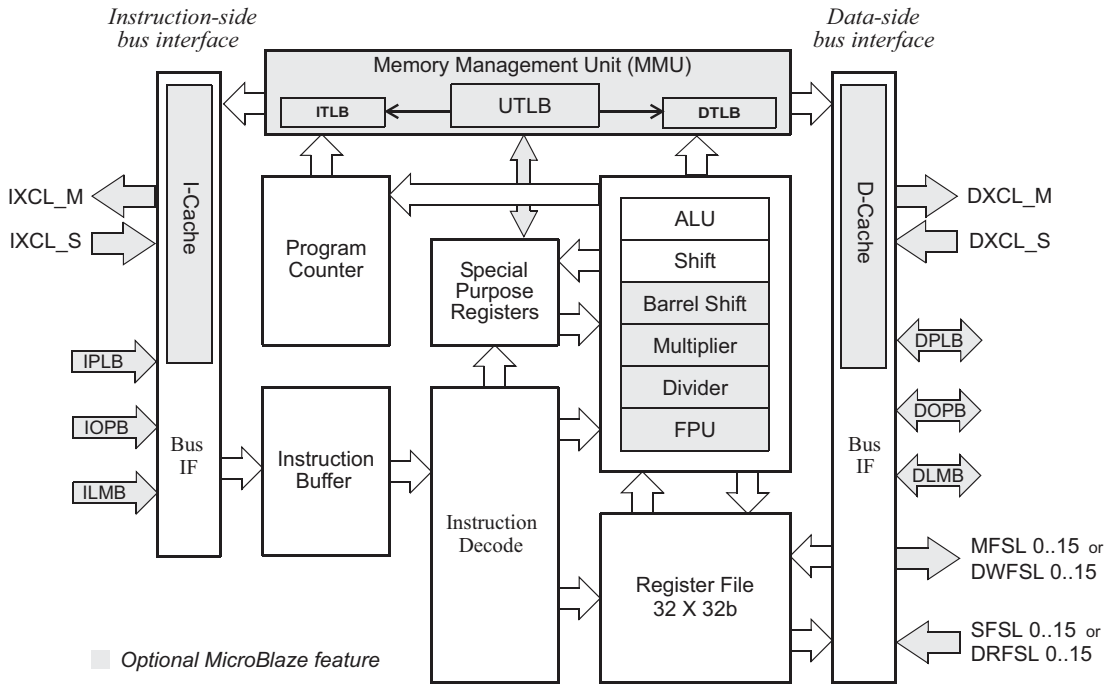


Figura B.2: Diagrama interno do processador MicroBlaze v7.10.a (retirado de [Xil08e]).

B.3.1.2 Registos Internos

O processador *MicroBlaze* possui 32 registos de uso geral e até 32 registos dedicados, dependendo das opções de configuração. Os 32 registos de uso geral são numerados de *R0* a *R31*.

À semelhança de outras arquitecturas, possui registos que devem ser utilizados de acordo com as convenções *caller-saved* e *called-saved*. O primeiro registo apresenta sempre o valor 0. Os registos *R14* a *R16* são utilizados para guardar os endereços de retorno de interrupções, vectores do utilizador e *breaks*, respectivamente.

Dos registos de uso específico, destacam-se:

- *Program Counter* - registo de 32 bits cujo conteúdo é o endereço da instrução actual em execução e pode ser acedido apenas para leitura;
- *Machine Status Register* - registo de estado do processador, possui a dimensão de 32 bits e internamente contém bits de estado e de controlo do processador;
- *Exception Address Register* - registo de endereço da excepção, guarda o endereço completo proveniente da instrução de acesso à memória que provocou a excepção;
- *Exception Status Register* - registo de estado da excepção, contém 32 bits que codificam qual a causa da excepção e os seus detalhes específicos.

Os restantes registos específicos são implementados opcionalmente, dependendo das funcionalidades requeridas para o sistema alvo, pelo que não serão abordados neste apêndice.

B.3.1.3 Arquitectura do Pipeline

A execução de instruções no processador *MicroBlaze* é *pipelined* e para a maior parte das instruções, cada etapa necessita de um ciclo de relógio para a sua conclusão. O mesmo será dizer que o número de ciclos de relógio para completar uma instrução concreta é igual ao número de etapas do *pipeline*, sendo completada uma instrução em cada ciclo de relógio. Algumas instruções requerem múltiplos ciclos de relógio na etapa de execução. Para garantir os requisitos temporais necessários, a execução do *pipeline* é suspensa (*pipeline stall*).

De acordo com o tipo de optimização escolhida, o *pipeline* do processador terá diferentes profundidades:

- *Pipeline* de três etapas - utilizado quando é activada a optimização de área, está dividido nas etapas: *Fetch*, *Decode* e *Execute*. O desempenho desta arquitectura foi avaliado pela *Xilinx* em *0.92 Dhrystone MIPS/MHz* [Xil08f].

- *Pipeline* de cinco etapas - utilizado quando é desactivada a optimização de área, sendo neste caso optimizado para desempenho, está dividido nas etapas: *Fetch*, *Decode*, *Execute*, *Access Memory* e *Writeback*. O desempenho desta arquitectura foi avaliado pela *Xilinx* em *1.15 DMIPS/MHz* [Xil08f].

Obviamente, a execução do *pipeline* com cinco etapas tem um acréscimo de desempenho à custa de maior área ocupada no dispositivo físico. Este é sem dúvida um dilema, no momento de desenvolvimento de um projecto onde área e desempenho são factores cruciais.

B.3.1.4 Arquitectura de Memória

O processador *MicroBlaze* foi desenhado com uma arquitectura de memória do tipo *Harvard*, isto é, com acesso independente às memórias de instruções e de dados. Cada endereço de memória é representado por 32 bits, o que permite até $4GB$ de espaço de endereçamento para cada tipo de acesso. Ambas as interfaces do processador são de 32 bits e utilizam o formato *Big-Endian*.

B.3.1.5 Fast Simplex Link

A versão actual do processador *MicroBlaze* suporta até 16 interfaces para ligações ponto-a-ponto unidireccionais de alta velocidade, designados *FSL - Fast Simplex Link*. Cada interface tem 32 bits de dimensão e é constituído por duas portas, uma de entrada e outra de saída. A comunicação é efectuada, por canais uni-direccionais do tipo *FIFO*, ponto-a-ponto dedicados.

O *FSL* proporciona comunicações de baixa latência, sendo bastante útil para interligar coprocessadores ao *MicroBlaze*. A arquitectura do processador *MicroBlaze* disponibiliza diversas instruções que permitem tirar partido das capacidades da interface *FSL*.

B.3.1.6 Depuração

O processador *MicroBlaze* inclui uma interface de depuração para suporte de ferramentas de *software* como a *XMD*. Este foi desenvolvido para conexões ao módulo *MDM - MicroBlaze Debug Module*, que proporciona a interface com o porto *JTAG* da *FPGA*.

As restantes funcionalidades suportadas pelo processador são facultativas e por não serem adequadas, ou desinteressantes do ponto de vista da avaliação do desempenho, não foram utilizadas na plataforma desenvolvida, pelo que não serão abordadas neste apêndice. Ainda no sentido de não tornar enfadonha a descrição do processador *MicroBlaze*, certos detalhes como o tratamento de interrupções, excepções, *user vector* e *breaks* não serão abordados neste apêndice.

B.4 O Processador PowerPC 405

O *PowerPC 405* é um processador *RISC* de *32* bits, dotado de uma arquitectura de memória do tipo *Harvard*. O processador *PowerPC 405* integrado nas *FPGAs Virtex-II Pro* está implementado em recursos dedicados (*hardwired*). Este é tipicamente utilizado sob a forma de um núcleo rígido que necessita de estar fisicamente implementado na *FPGA* alvo. Desenvolvido em 1998 [IBM08b], possui um desempenho avaliado pela *IBM* em *1.52 DMIPS/MHz* [IBM08c]. Para a *FPGA* utilizada pode operar à frequência máxima de $400MHz$. As principais funcionalidades suportadas são:

- Unidade de gestão de memória (*MMU*) - possibilita a execução do *software* de forma mais segura e robusta;
- Hierarquia de interrupções com dois níveis de prioridade - permite a distinção entre interrupções críticas e não críticas;
- Memória *cache* integrada - reduz a latência na obtenção do conteúdo residente em memória externa;
- Suporte de unidades de processamento auxiliares (*APUs*) - possibilita a adição de co-processadores e a extensão do conjunto de instruções do processador com instruções a serem executadas em coprocessadores dedicados;
- Processador *pipelined* - cinco etapas de *pipeline* que permite, para a maioria das instruções, concluir uma instrução em cada ciclo de relógio [Xil08j].

B.4.1 Arquitectura do Processador

A figura B.3 ilustra a arquitectura interna do processador *PowerPC 405*.

B.4.1.1 Tipos de Dados e sua Formatação

O processador *PowerPC 405* utiliza por omissão o formato *Big-Endien* para representação de dados. Suporta os seguintes tipos de dados:

- `Quadword` - 128 bits de dimensão;

- `Doubleword` - 64 bits de dimensão;
- `Word` - 32 bits de dimensão;
- `Halfword` - 16 bits de dimensão;
- `Byte` - 8 bits de dimensão.

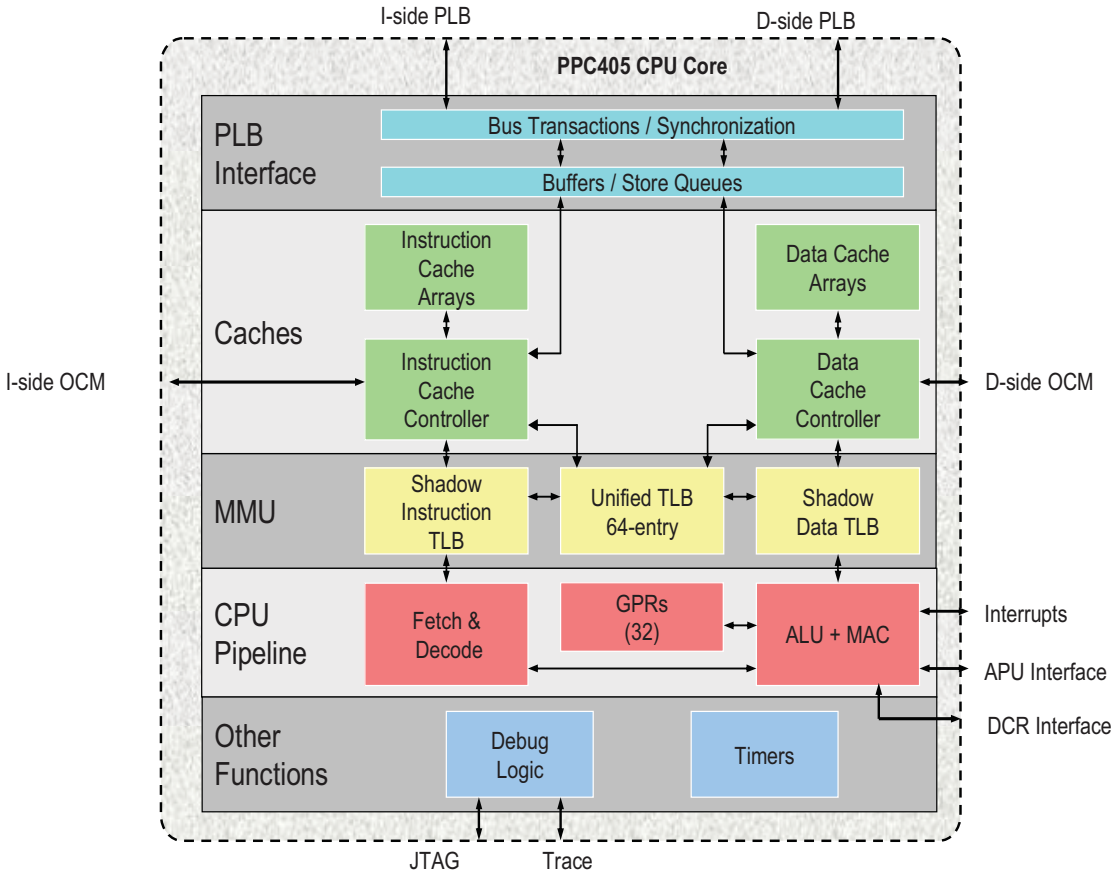


Figura B.3: Diagrama interno do processador PowerPC 405 (retirado de [IBM08d]).

B.4.1.2 Registos Internos

O processador *PowerPC 405* possui 32 registos de uso geral, numerados de *R0* a *R31* e 32 registos dedicados. Estes últimos permitem acesso adicional a recursos do processador como o registo de estado, registos de excepções e registos dos temporizadores.

Nesta plataforma de desenvolvimento, o processador *PowerPC 405* é regido pela interface de programação *EABI* (*Embedded Application Binary Interface*) . Este especifica um conjunto de convenções para aplicações embutidas e foi definido para garantir compatibilidade para um conjunto de funcionalidades descritas [IBM08a].

Os registos de uso geral estão convencionados na interface *EABI*, este define registos *caller-saved* e *called-saved*, registos de passagem de parâmetros e de retorno de valores, registo para

o *stack pointer* e registos para acesso de leitura e de escrita a áreas de dados pequenas (*small data area anchor*). A utilização da *stack* está também convencionada na interface *EABI*. Este define a estrutura *stack frame*, onde especifica os registos que o cabeçalho de cada *frame* deve conter, as zonas para a passagem de parâmetros e de variáveis locais e a ordem pela qual os registos do processador devem ser salvaguardados.

B.4.1.3 Arquitectura de Memória

O processador *PowerPC 405* foi desenhado com uma arquitectura de memória do tipo *Harvard*, onde é separado o acesso a instruções e dados. Cada endereço de memória é representado por 32 bits, o que permite até 4GB de espaço de endereçamento para cada tipo de acesso. A arquitectura de memória do *PowerPC 405* suporta a ordenação de informação (*endianness*) nos formatos *Big-Endian* e *Little-Endian*. O processador possui ainda o registo dedicado *Storage Little-Endian Register* (*SLER*), que permite discriminar a ordenação de informação para as diversas regiões de memória.

B.4.1.4 Interface Processor Local Bus

A interface *Processor Local Bus* (*PLB*) do processador dispõe de um barramento de endereços de 32 bits e três barramentos de dados de 64 bits, ligados às unidades de *cache* de dados e de instruções [Xil08i]. Dos três barramentos de dados, dois estão ligados à unidade de *cache* de dados onde um é dedicado para operações leitura e outro para operações de escrita. Por fim, o terceiro barramento de dados está ligado à unidade de *cache* de instruções.

De referir ainda que os acessos a memórias externas passam necessariamente pelo barramento *PLB*, pelo que desempenho global do sistema é influenciado pela frequência de funcionamento do barramento e ainda pelo número de *slaves* conectados ao barramento.

B.4.1.5 Cache

Actualidade, o elevado desempenho dos sistemas computacionais é em grande parte devido ao uso de unidades de *cache*. Estas funcionam como elo de ligação entre o processador e a memória do sistema. A estas unidades cabe minimizar a latência do acesso à informação contida na memória do sistema. A *cache* contém cópias dos dados e das instruções mais frequentemente utilizadas e tipicamente pode ser acedida de forma bastante mais rápida que a memória do sistema.

O processador *PowerPC 405* possui duas unidades de *cache* físicas, dedicadas para instruções e dados respectivamente. Cada unidade de *cache* possui $16KB$ de capacidade e dois níveis de associatividade.

B.4.1.6 Depuração

O processador *PowerPC 405* permite depuração em modo interno, externo, *debug-wait* e tempo-real [Xil08i]. No modo interno, a depuração pode ser efectuada por intermédio de ferramentas de *software*, sendo possível analisar o *software* do sistema e os programas da aplicação. O modo externo é utilizado por *debuggers JTAG* e permite analisar *hardware* e *software*. O modo *debug-wait* é idêntico ao anterior, com a nuance das interrupções externas serem atendidas mesmo quando o processador está parado para depuração. Finalmente, no

modo de tempo-real os eventos de depuração são utilizados para sinalizar às ferramentas externas os instantes em que estas devem recolher informação. Refira-se ainda que este modo não altera o desempenho do processador.

As restantes funcionalidades suportadas pelo processador não são interessantes do ponto de vista da avaliação do desempenho e não foram utilizadas na plataforma desenvolvida, pelo que não serão abordadas neste apêndice. Ainda no sentido de não tornar enfadonha a descrição do processador *PowerPC 405*, certas funcionalidades como a unidade de gestão de memória, o tratamento de interrupções e de excepções não serão abordados neste apêndice.

B.5 Fluxo de Projecto

Para possibilitar a implementação e a comparação do executivo *OReK* entre diferentes arquitecturas, foi necessário criar duas novas plataformas, denominadas por *MB-SoC* e *PPC-SoC*, respectivamente. Estas foram desenvolvidas com o auxílio das ferramentas acima descritas. A sua prototipagem foi realizada numa *FPGA XC2VP30-7ff896* da família *Virtex-II Pro* da *Xilinx* [Dig08].

A figura B.4 ilustra o fluxo de projecto simplificado para um sistema de tempo-real embutido baseado nas plataformas de *hardware* e de *software*. De notar que esta figura não apresenta as etapas de desenvolvimento do modelo dos sistemas integrados *MB-SoC* e *PPC-SoC* nem do respectivo *software* de suporte, concentrando-se apenas na implementação das aplicações.

As metades esquerda e direita da figura B.4 ilustram os subfluxos de *hardware* e de *software*, respectivamente. A plataforma de implementação/prototipagem é mostrada na parte inferior da figura B.4, consistindo numa placa "Xilinx University Program Virtex-II Pro" comercializada pela *Digilent* [Dig08]. Além da *FPGA XC2VP30-7ff896* da *Xilinx*, a placa possui uma memória *Flash* para armazenamento não volátil de configurações da *FPGA*, memórias *DDR SDRAM* e *Compact Flash*, e um conjunto de componentes auxiliares, tais como reguladores, osciladores, controladores, *drivers* de linha e fichas para ligação a dispositivos externos.

B.5.1 O Subfluxo de Hardware

As próximas subsecções são relativas às plataformas desenvolvidas no contexto deste trabalho, nomeadamente os sistemas integrados *MB-SoC* e *PPC-SoC*.

B.5.1.1 Implementação da plataforma MB-SoC

A plataforma de *hardware MB-SoC* integra os seguintes módulos:

- Processador *MicroBlaze*;
- Memória interna;
- Módulos *GPIO*;
- Módulo de depuração;
- *UART*;

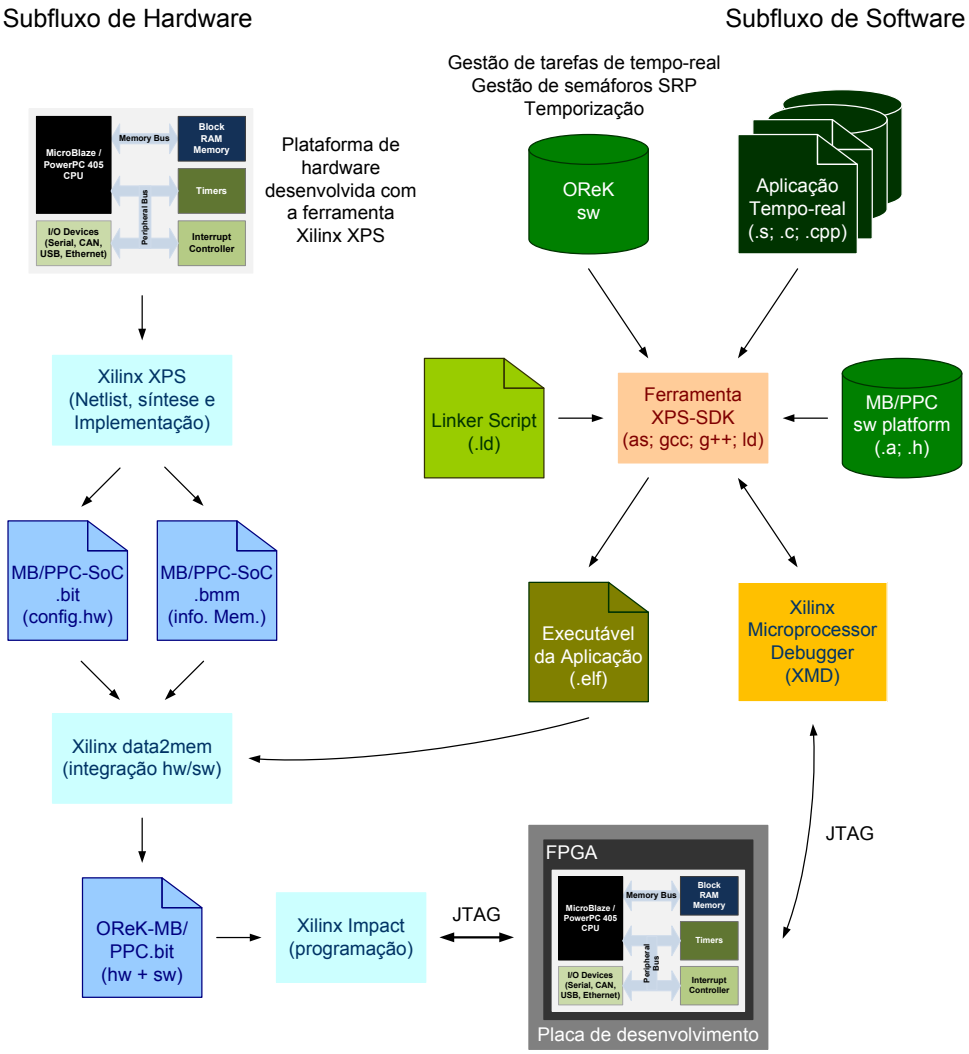


Figura B.4: Fluxo de projecto simplificado para aplicações baseadas nos sistemas integrados *OReK-MB/OReK-PPC*.

- Controlador de interrupções;
- Temporizador;
- Módulo de gestão digital de clock (*DCM*);
- Memória *DDR*;
- Módulo de temporizadores para avaliação de desempenho;
- Barramentos *OPB*, *LMB* e *FSL*;
- Memória interna ligada por *OPB*.

Por questões internas da versão da plataforma de *software* utilizada (*Xilinx EDK 9.2.02i*), a última versão do processador *MicroBlaze* suportado para *FPGAs* da familia *Virtex-II* é a `6.00.b`. As consequências mais imediatas são a perda das funcionalidades introduzidas no último processador disponibilizado na plataforma de *software*, o *MicroBlaze* `7.00.b`. Mais concretamente, a unidade de gestão de memória *MMU* e a interface para o barramento *PLB* (*Processor Local Bus*).

O processador implementado foi configurado com as seguintes características: optimização de área desactivada, *barrel shifter* activado , multiplicação e divisão de inteiros activadas, unidade de vírgula flutuante desactivada, *pattern comparator* e módulo de depuração activado.

Nesta implementação pretendeu-se desenvolver e avaliar um sistema com características determinísticas, pelo que as funcionalidades de *cache* foram desactivadas.

Ainda relativamente ao processador implementado, foi especificada a frequência de funcionamento de *110 MHz*.

A memória interna implementada na plataforma é do tipo *BRAM*, interligada ao processador através do barramento *LMB* (*Local Memory Bus*). Foram utilizados dois módulos com *64 KB* para armazenamento de dados e de instruções, respectivamente.

A memória implementada na plataforma e ligada ao processador via *OPB* é do tipo *BRAM*. Foi utilizado um módulo com *64* KB, interligado ao processador através do barramento *OPB* (*On-chip Peripheral Bus*).

Os módulos *GPIO* implementam a interface para os portos de entrada e de saída do dispositivo físico, nomeadamente: botões de pressão, botões de posição e *LEDs*. Utilizou-se o barramento *OPB* para interligação entre estes e o processador. Os módulos *GPIO* dos respectivos portos de entrada foram ainda ligados ao controlador de interrupções.

O módulo de depuração implementado foi o *MDM*, motivado pelo suporte garantido pelas ferramentas de *software* da *Xilinx*. Este foi interligado ao processador por intermédio dos barramentos *OPB* e *FSL*.

A *UART* implementada utiliza apenas as funcionalidades essenciais (*UART lite*) e possui interface para ligação ao barramento *OPB*. O módulo foi ainda configurado para um *baud*

rate de *115200*, com *8* bits de dados e sem paridade.

O controlador de interrupções implementado permite até *32* fontes de interrupção geridas por um vector de prioridades, isto é, a prioridade de cada interrupção é dada pela sua posição no vector. O módulo foi interligado ao processador por intermédio de uma linha de interrupção e do barramento *OPB*.

O módulo de temporização implementado foi configurado para suportar internamente dois temporizadores/contadores, cada um possui a dimensão de *32* bits. Este foi interligado à primeira entrada (fonte de interrupção) do controlador de interrupções. A conexão ao processador por obtida por intermédio do barramento *OPB*.

De modo a possibilitar a manipulação do sinal de relógio do dispositivo físico, foi introduzido um *DCM*, configurado para gerar um sinal de relógio com a frequência de *110 MHz*. Este sinal de relógio foi interligado a todos os módulos da plataforma, ditando a sua frequência de funcionamento.

Foi ainda adicionado um módulo de temporizadores para avaliação de desempenho. Este módulo foi desenvolvido no âmbito desta dissertação com a finalidade de proporcionar uma ferramenta de avaliação precisa. O módulo possui uma interface *FSL* que possibilita a sua interligação ao processador como uma unidade de co-processamento. Desta forma é minimizada a latência e interferência do módulo de avaliação no executivo *OReK*. O apêndice C aborda de forma mais detalhada o módulo desenvolvido.

Posteriormente foi adicionado suporte para uma memória externa do tipo *DDR*, com 256 *MB* de capacidade e 266 *MHz* de frequência de funcionamento. Esta nova funcionalidade requereu a adição de um módulo controlador de memória *DDR*, ligado ao processador através do barramento *OPB* e de um novo sinal de relógio para ligação à memória *DDR*. A inclusão da memória externa impôs uma nova frequência de funcionamento de toda a plataforma de *hardware*. Esta passa a ter o valor de *100 MHz*, sendo necessária uma re-configuração do dispositivo *DCM* para gerar um sinal com a frequência especificada.

B.5.1.2 Implementação da plataforma PPC-SoC

A plataforma de *hardware PPC-SoC* integra os seguintes módulos (ver figura B.5):

- Processador *PowerPC 405*;
- Memória interna;
- Módulos *GPIO*;
- Módulo de depuração;
- *UART*;
- Controlador de interrupções;
- Temporizador;

- Módulo de gestão digital de clock (*DCM*);
- Memória *DDR*;
- Módulo de temporizadores para avaliação de desempenho;
- Barramentos *OPB*, *DOCM*, *IOCM* e *PLB*.

A plataforma de *hardware* foi desenvolvida na versão 10.1 da ferramenta *Xilinx EDK*. O núcleo de propriedade intelectual do processador *PowerPC 405* foi implementado na versão 2.00.c e foi configurado com as seguintes características: *forwarding* do operando para instruções de carregamento de informação desactivado, unidade de gestão de memória activada e re-sincronização *DCR* desactivada.

Ainda relativamente ao processador implementado, foi especificada a frequência de funcionamento de *300 MHz*. De referir que tal frequência de funcionamento só é aplicável ao processador por este se tratar de um núcleo implementado em lógica dedicada e interno à *FPGA*. Os restantes módulos são de origem sintetizável pelo que a frequência de funcionamento é necessariamente inferior. Para o projecto desenvolvido, a frequência de funcionamento utilizada nos restantes módulos foi de *100 MHz*.

A memória interna implementada na plataforma é do tipo *BRAM*, ligada ao processador através dos barramentos *DOCM* (*Data Side On-Chip Memory Bus*) e *IOCM* (*Instruction Side On-Chip Memory Bus*), para acesso de dados e de instruções, respectivamente.

Foram utilizados dois módulos de memória com a dimensão de *64 KB* e *128 KB* para acesso de dados e de instruções, respectivamente.

O módulo de depuração implementado foi o *PowerPC JTAG Controller*, motivado pelo seu suporte estar assegurado pelas ferramentas de *software* da *Xilinx*. Foi interligado ao processador por intermédio do barramento *JTAG*.

A *UART*, o controlador de interrupções, o módulo de temporização e os diversos módulos *GPIO* implementados possuem uma configuração igual à utilizada na implementação da plataforma *MB-SoC*. Para o efeito foi adicionado um módulo de interligação de barramentos (*bridge*), que possibilita a comunicação entre os barramentos *PLB* e *OPB*.

O módulo *DCM* foi configurado para gerar dois sinais de relógio com as frequências *300 MHz* e *100 MHz*. O primeiro sinal foi ligado ao processador e o segundo aos restantes módulos da plataforma.

Foi ainda adicionado suporte para uma memória externa do tipo *DDR*, com *256 MB* de capacidade e *266 MHz* de frequência de funcionamento. Esta nova funcionalidade requereu a adição de um módulo controlador de memória *DDR*, ligado ao processador através do barramento *PLB* e de um novo sinal de relógio para ligação à memória *DDR*.

Por fim, o módulo de temporizadores para avaliação de desempenho foi interligado à plataforma de *hardware* através do barramento *PLB*. O apêndice C aborda de forma mais detalhada o módulo desenvolvido.

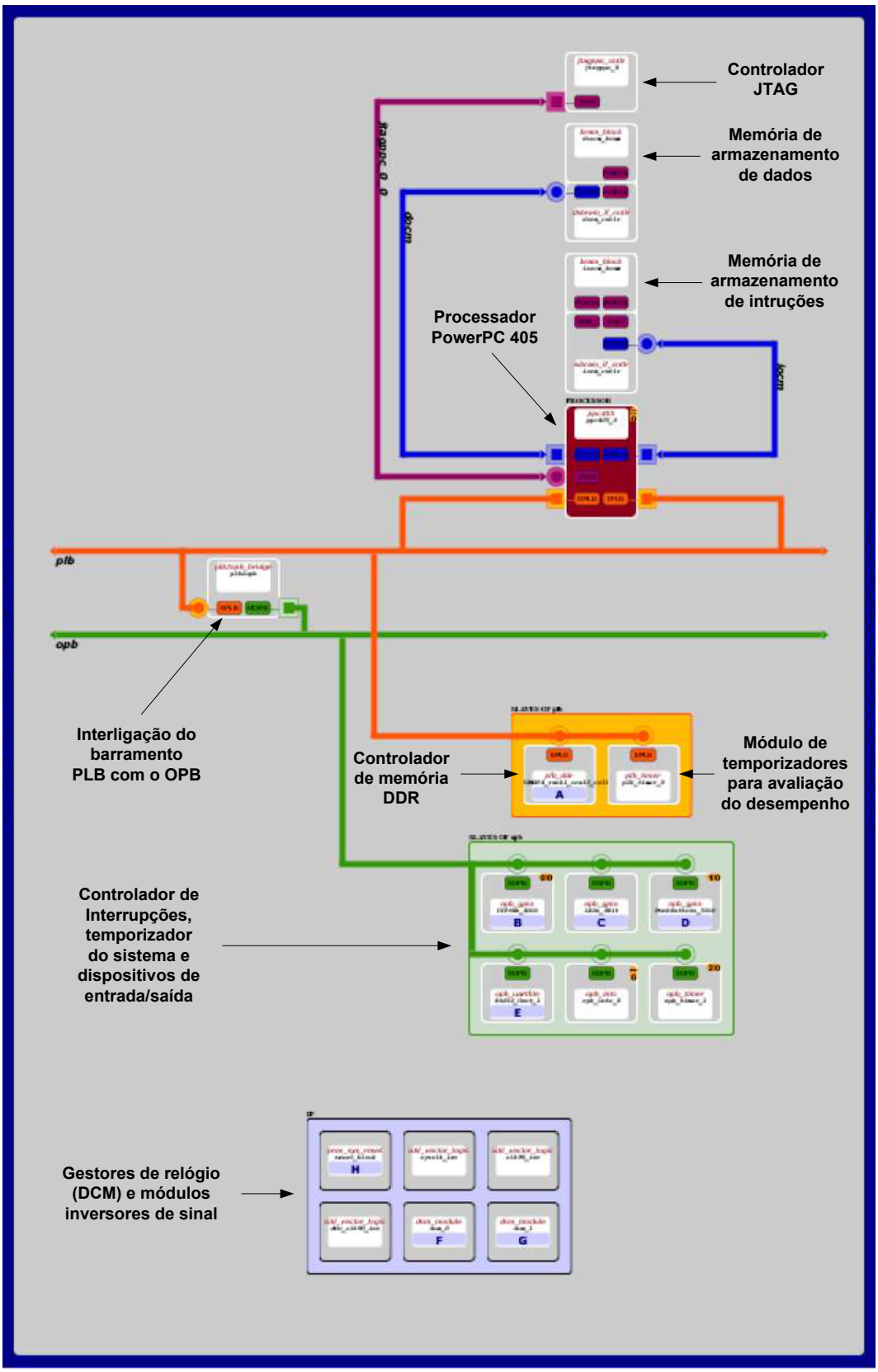


Figura B.5: Diagrama interno do sistema integrado *PPC-SoC* (extraído da ferramenta de desenvolvimento *XPS*).

B.5.2 O Subfluxo de Software

Esta subsecção descreve de forma sucinta as funções necessárias para o correcto funcionamento do executivo *OReK* nos sistemas integrados *MB-SoC* e *PPC-SoC*.

O desenvolvimento da plataforma de *software* consistiu na criação de um projecto de *C++* para possibilitar o porte e posterior avaliação de desempenho do executivo *OReK*. Nesse sentido, foram desenvolvidas as seguintes funções de baixo nível e dependentes da plataforma:

- `void StartSystemTimer(uint baseFrequency, uint timeResolution, TTimerCallback timerCallback);`
- `void StopSystemTimer(void);`
- `void UnmaskInterrupt(uchar intNumber);`
- `void MaskInterrupt(uchar intNumber);`
- `void MaskAllInterrupts();`
- `void RegIntCallback(TIntCallback functPtr);`
- `void GenerateTimerInterrupt();`
- `void TimerHandler(void*);`
- `void InterruptHandler(void* intNumber);`
- `void OnTaskEnd(uchar contxtNum, uint stackPointer);`
- `void _interrupt_handler(void);`
- `OReK_non_critical_interrupt(base, int_num)`.

A função `StartSystemTimer` é responsável pela activação do temporizador do executivo *OReK*. Esta programa o temporizador da plataforma de *hardware* para gerar interrupções periódicas, com a resolução temporal especificada pelo utilizador. Além disso, regista a função `TimerHandler` para que a sua invocação seja feita no contexto da rotina de serviço à interrupção (*RSI*) do temporizador. A função `StartSystemTimer` foi escrita em linguagem *C* e possui directivas do pré-processador para inclusão condicional de código exclusivo da arquitectura. Esta metodologia permite o funcionamento da função `StartSystemTimer` nas plataformas *MB-SoC* e *PPC-SoC*.

A função `StopSystemTimer`, tal como o nome sugere é responsável pela desactivação do temporizador do executivo *OReK*. A função foi escrita em linguagem *C* e opera nas plataformas *MB-SoC* e *PPC-SoC*.

As funções `UnmaskInterrupt`, `MaskInterrupt` e `MaskAllInterrupts`, actuam unicamente no dispositivo controlador de interrupções e são responsáveis por: activar uma fonte de interrupção, desactivar uma fonte de interrupção e desactivar todas as fontes de interrupção à excepção da interrupção do temporizador, respectivamente. As funções foram escritas em linguagem *C* e operam nas plataformas *MB-SoC* e *PPC-SoC*.

A função `RegIntCallback` é responsável por registar a função `InterruptHandler` de forma a que a sua invocação seja feita no contexto da *RSI* do controlador de interrupções. A função `InterruptHandler` deve ser invocada para todas as fontes de interrupção do controlador de interrupções, à excepção do temporizador do executivo *OReK*. A função foi escrita em linguagem *C* e opera nas plataformas *MB-SoC* e *PPC-SoC*.

A função `GenerateTimerInterrupt` é responsável pela activação de um temporizador, incluído no módulo de temporização da plataforma de *hardware*. O temporizador é ainda configurado para gerar uma interrupção logo após o instante de activação. O objectivo desta função é gerar pseudo-interrupções despoletadas por *software*. A função foi escrita em linguagem *C* e opera nas plataformas *MB-SoC* e *PPC-SoC*.

As funções `TimerHandler` e `InterruptHandler`, são invocadas no contexto de *RSIs* ao evento gerado pelo controlador de interrupções. Estas invocam as funções `TimerCallback` e `IntCallback` do executivo *OReK*, respectivamente. As funções `TimerHandler` e `InterruptHandler` são ainda responsáveis por sinalizar que as respectivas rotinas de serviço foram atendidas. Ambas as funções foram escritas em linguagem *C* e operam nas plataformas *MB-SoC* e *PPC-SoC*.

A função `OnTaskEnd` tem como funcionalidade a troca do ponteiro da pilha (*stack*) e posterior invocação da função `GenTaskEndException`. A função foi escrita em linguagem *assembly* específica dos processadores de ambas plataformas. Devido à necessidade da sua implementação ser efectuada de forma segura e em linguagem de baixo nível, são disponibilizadas duas versões para o funcionamento nas plataformas *MB-SoC* e *PPC-SoC*.

A função `_interrupt_handler` é responsável pelo atendimento de interrupções do processador *MicroBlaze*. Esta implementa a salvaguarda e o restauro do contexto de execução do processador e invoca a rotina de serviço do dispositivo que gerou a interrupção no processador. De salientar que é nesta função que a comutação de contexto do executivo *OReK* na plataforma *MB-SoC* é efectuada. A função foi escrita em linguagem *assembly* do processador *MicroBlaze* e opera unicamente na plataforma *MB-SoC*.

Finalmente, a função `OReK_non_critical_interrupt` é responsável pelo atendimento de interrupções do processador *PowerPC 405*. Esta implementa a salvaguarda e o restauro do contexto de execução do processador de acordo com a especificação *EABI* [IBM08a] e invoca a rotina de serviço do dispositivo que gerou a interrupção no processador. De salientar que é nesta função que a comutação de contexto do executivo *OReK* na plataforma *PPC-SoC* é efectuada. A função foi escrita em linguagem *assembly* do processador *PowerPC 405* e opera unicamente na plataforma *PPC-SoC*.

Apêndice C

Temporizadores para Avaliação do Desempenho

C.1 Introdução

Este apêndice apresenta o módulo de temporizadores desenvolvidos, com a finalidade de proporcionar uma avaliação temporal precisa, dos sistemas embutidos *OReK-MB* e *OReK-PPC*.

O módulo foi desenvolvido em *VHDL*, sendo sintetizável e independente da tecnologia de implementação alvo, embora a sua interface deva ser com um dos processadores utilizados. A sua prototipagem foi realizada numa *FPGA XC2VP30-7ff896* da família *Virtex-II Pro* da *Xilinx* [Dig08].

C.2 Características Gerais

Na fase de projecto do módulo de temporizadores, foram especificadas as seguintes características:

- 5 contadores/temporizadores independentes;
- dimensão de 64 bits para cada contador;
- possibilidade de especificar um valor inicial para os diversos contadores;
- possibilidade de activar e suspender contadores, de forma individual ou simultânea;
- capacidade de leitura do valor dos contadores activos e suspensos;
- interface *FSL* para possibilitar comunicação de baixa latência com o processador *MicroBlaze*;
- Interface *PLB* para possibilitar comunicação com o processador *PowerPC 405*.

C.3 Projecto do Hardware

No sentido de desenvolver o módulo com as características acima descritas, foi necessário projectar os diversos contadores e uma máquina de estados. Esta proporciona o controlo interno do módulo de temporizadores e implementa a interface *FSL* [Xil08c].

Posteriormente foi adicionado a interface *Processor Local Bus* (*PLB*) [Xil08k]. Esta actualização, apesar de implementar um protocolo de comunicação que requer mais recursos do dispositivo físico e maior latência que o barramento *FSL*, tem a vantagem de ser mais genérico. Actualmente este é o barramento de comunicação utilizado nos processadores *MicroBlaze* e *PowerPC 405* da plataforma *Xilinx EDK*.

O projecto foi sintetizado para a *FPGA XC2VP30-7ff896*, com a ferramenta *Xilinx ISE 9.2.04i* [Xil08m]. Refira-se ainda que a versão implementada com a interface *FSL* ocupa aproximadamente *500 slices* e opera a uma frequência máxima de *180 MHz*.

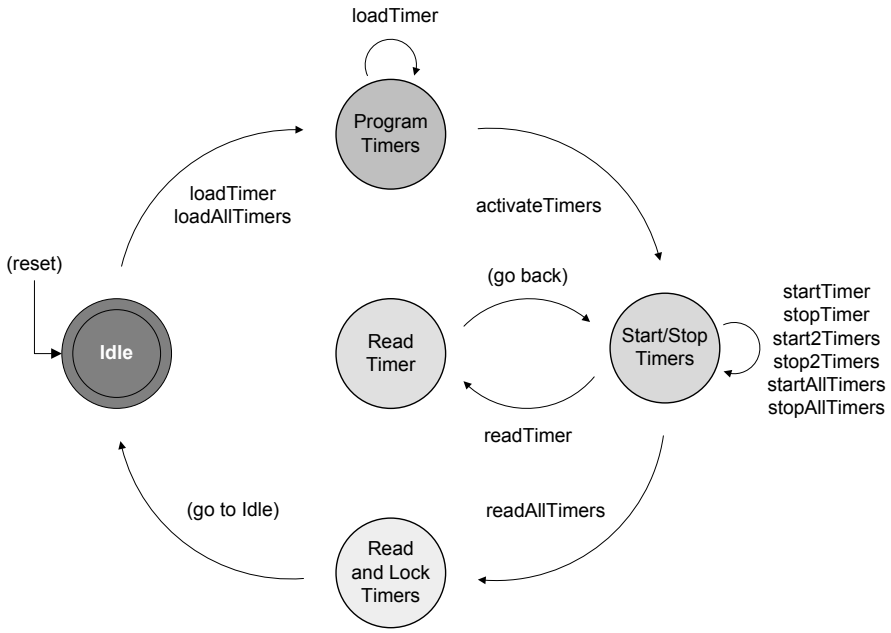


Figura C.1: Diagrama de estados e funções de software do módulo de temporizadores.

A figura C.1 mostra de uma forma simplificada, a máquina de estados anteriormente referida. Nesta são ilustrados os diversos estados do módulo de temporizadores e as funções de *software* desenvolvidas. Após o *reset* do sistema computacional, a máquina de estados é colocada num estado seguro (*Idle*). Este inicializa todos os contadores a zero e aguarda informação de controlo proveniente do processador. De acordo com a implementação utilizada, a informação provém exclusivamente do barramento *FSL* ou do barramento *PLB*. Como consequência da chegada de informação, é efectuada uma transição para o estado de configuração do módulo de temporizadores.

O estado de configuração (*Program Timers*) valida a informação proveniente do processador. De acordo com o seu conteúdo, esta é utilizada para definir o valor inicial para os diversos contadores do módulo.

Após a inicialização dos contadores pretendidos é efectuada uma transição para o estado *Start/Stop Timers*. Este permite efectuar de forma completamente arbitrária, o arranque e pausa dos diversos temporizadores que o módulo dispõe. Mais uma vez, o seu controlo é efectuado pela validação de informação proveniente do processador. Além do controlo individual dos diversos temporizadores, este estado possibilita a actuação simultânea sobre pares de temporizadores ou globalmente, para todos os temporizadores. Desta forma é possível o arranque (ou pausa) simultânea de diversos contadores, particularmente útil para avaliar temporalmente eventos de forma simultânea e ainda para diminuir a sobrecarga e intrusão das funções de controlo do módulo de temporizadores.

Para possibilitar a leitura do valor de um temporizador em execução (*run-time*), foi introduzido um estado secundário (*Read Timer*). Este permite a consulta dos diversos temporizadores, sem afectar o seu funcionamento. Após a leitura do temporizador desejado, o estado volta para o controlo dos temporizadores. Desta forma, o módulo desenvolvido permite o controlo e consulta de cada temporizador, de forma completamente arbitrária.

Finalmente, quando for desejada a terminação com consulta de todos os temporizadores, é efectuada uma nova transição de estado. Neste último estado, todos os temporizadores são parados em simultâneo, seguindo-se o envio para o processador do conteúdo de todos os temporizadores e a transição para o estado inactivo.

C.4 Interface de Software

Para tirar partido das potencialidades do módulo projectado, foi desenvolvida uma camada de *software* de baixo nível, tipicamente denominada *device driver*. Este deve ser utilizado na aplicação que em se pretende efectuar as medições temporais, pelo que foi projectado para possibilitar uma utilização completa, mas ao mesmo tempo, simples e intuitiva de todas as funcionalidades disponibilizadas pelo módulo de temporizadores. Ainda relativamente ao *driver*, foram desenvolvidas duas versões: ambas implementam as macros ilustradas na figura C.2 porém, a primeira interage com o módulo de *hardware* através do barramento *FSL*, enquanto que a segunda transfere informação via *PLB*.

Uma vez que foram projectados para minimizar o seu impacto na execução da aplicação, os *drivers* desenvolvidos foram escritos na forma de macros, cuja principal vantagem é a eliminação da sobrecarga computacional provocada pelas instruções de salto, salvaguarda e restauro de contexto do processador. À excepção das macros de inicialização e das macros de leitura dos temporizadores, as restantes macros podem ser codificadas em apenas duas ou três instruções em linguagem *assembly*, de acordo com a interface de comunicação implementado *FSL* ou *PLB*, respectivamente.

As macros `loadTimer` e `loadAllTimers`, permitem definir o valor inicial para os diversos contadores. Os três parâmetros da primeira macro permitem identificar o contador, os 32 bits mais significativos e os 32 bits menos significativos do contador, respectivamente. A cada execução da macro, é configurado o valor inicial do contador especificado. Para configurar todos os contadores com o mesmo valor inicial, deve-se recorrer à macro `loadAllTimers`, esta atribui o conteúdo dos seus dois parâmetros a todos os contadores do módulo. Ambas as

```
loadTimer(timerId, startupValHi, startupValLo)
loadAllTimers(startupValHi, startupValLo)

activateTimers()

startTimer(timerId)
start2Timers(timer1Id, timer2Id)
startAllTimers()

stopTimer(timerId)
stopAndLockTimer(timerId)
stop2Timers(timer1Id, timer2Id)
stopAllTimers()

readTimer(timerId, timerHi, timerLo)
readAllTimers(timer4Hi, timer4Lo,    \
              timer3Hi, timer3Lo,    \
              timer2Hi, timer2Lo,    \
              timer1Hi, timer1Lo,    \
              timer0Hi, timer0Lo)
```

Figura C.2: Protótipos das macros desenvolvidas para controlo do módulo de temporizadores.

macros são de utilização facultativa, sendo necessárias quando se pretende a inicialização dos contadores com valores diferentes de zero.

A macro `activateTimers` provoca a transição para o estado de arranque/pausa dos temporizadores. Isto é, conclui a inicialização do módulo de temporizadores e transita para o estado seguinte. Ao contrário das anteriores, esta macro é de utilização obrigatória.

As macros `startTimer`, `start2Timers` e `startAllTimers` permitem iniciar a contagem crescente de um contador, dois contadores ou todos os contadores, respectivamente. Os parâmetros passados nas primeiras duas macros especificam quais os contadores que devem entrar em funcionamento. Utilizando uma das últimas duas macros é possível iniciar simultaneamente vários contadores do módulo, o que possibilita uma latência nula entre a activação dos contadores que vão entrar em funcionamento.

As macros `stopTimer`, `stop2Timers` e `stopAllTimers` permitem suspender a contagem de um contador, dois contadores ou todos os contadores, respectivamente. Os parâmetros passados nas primeiras duas macros especificam quais os contadores que devem ser suspendidos. De forma análoga às macros anteriormente referidas é possível suspender simultaneamente vários contadores do módulo.

Finalmente, as macros `readTimer` e `readAllTimers` possibilitam a leitura de um contador ou leitura de todos os contadores do módulo, respectivamente. Na primeira é necessário especificar qual o contador que se pretende ler. É feita uma consulta do conteúdo do contador, podendo este encontrar-se suspendido ou em funcionamento. A macro `readAllTimers` força a paragem simultânea de todos os contadores, espera que o conteúdo de todos os contadores seja lido e posteriormente força a transição para o estado *idle*.

Apêndice D

Lista de Acrónimos

ADC - Analog-to-Digital Converter

ANSI - American National Standards Institute

API - Application Programmable Interface

ARM - Advanced RISC Machine

ARPA - Advanced Real-time Processor Architecture

ARPA-CP - ARPA with Configurable Pipeline

ARPA-MT - ARPA Multithreaded

ARPA-OSC - ARPA Operating System Coprocessor

ARPA-SoC - ARPA System-on-Chip

ASIC - Application Specific Integrated Circuit

APU - Auxiliary Processing Unit

BRAM - Block RAM

BSB - Base System Builder

BSP - Board Support Package

CAD - Computer Aided Design

CAN - Controller Area Network

CISC - Complex Instruction Set Computer

CLB - Configurable Logic Block

Cop0-MEC - Coprocessor 0 - Memory and Exceptions Coprocessor

Cop2-OSC - Coprocessor 2 - Operating System Coprocessor

CPI - Cycles per Instruction

CPU - Central Processing Unit

DAC - Digital-to-Analog converter

DCM - Digital Clock Manager

DDR - Double Data Rate

DM - Deadline Monotonic

DMA - Direct Memory Access

DMIPS - Dhrystone MIPS

DOCM - Data Side On-Chip Memory

DRAM - Dynamic Random Access Memory

DSP - Digital Signal Processor

EABI - Embedded Application Binary Interface

EDF - Earliest Deadline First

EDK - Embedded Development Kit

EEMBC - Embedded Microprocessor Benchmark Consortium

EEPROM - Electrically Erasable and Programmable Read Only Memory

FCFS - First Come - First Served

FIFO - First In - First Out

FIT - Fixed Interval Timer

FPGA - Field Programmable Gate Array

FPLD - Field Programmable Logic Device

FPU - Floating Point Unit

FSL - Fast Simplex Link

FTP - File Transfer Protocol

GPIO - General Purpose Input/Output

HDL - Hardware Description Language

IIC - Inter-Integrated Circuit

IOCM - Instruction Side On-Chip Memory

IP - Intellectual Property

IP - Internet Protocol

ISE - Integrated Synthesis Environment

ISP - In System Programmable

JTAG - Joint Test Action Group

LCD - Liquid Cristal Display

LMB - Local Memory Bus

LSb - Least Significant bit

LSF - Least Slack First

LUT - Lookup Table

MB-SoC - MicroBlaze - System-on-Chip

MDM - Microprocessor Debug Module

MDM - MicroBlaze Debug Module

MIPS - Microprocessor without Interlocked Pipeline Stages

MIPS - Millions of Instructions per Second

MIT - Minimum Interarrival Time

MMU - Memory Management Unit

MPU - Memory Protection Unit

MSb - Most Significant bit

MSDOS - Microsoft Disk Operating System

NMI - Non Maskable Interrupt

NoC - Network-on-Chip

OOP - Object Oriented Paradigm

OPB - On-chip Peripheral Bus

OReK - Object oriented Real-time Kernel

PC - Program Counter

PC - Personal Computer

PCI - Peripheral Component Interconnect

PCP - Priority Ceiling Protocol

PIP - Priority Inheritance Protocol

PLB - Processor Local Bus

POSIX - Portable Operating System Interface

PPC-SoC - PowerPC 405 - System-on-Chip

RAM - Random Access Memory

RISC - Reduced Instruction Set Computer

RM - Rate Monotonic

RR - Round-Robin

RT - Real-time

RTL - Register Transfer Level

RTOS - Real-time Operating System

SCB - Semaphore Control Block

SDK - Software Development Kit

SDRAM - Synchronous Dynamic RAM

SMT - Simultaneous Multithreading

SNMP - Simple Network Management Protocol

SoC - System-on-Chip

SPI - Serial Peripheral Interface

SRAM - Static Random Access Memory

SRP - Stack Resource Policy

TCB - Task Control Block

TCP - Transmission Control Protocol

TLB - Translation Lookaside Buffer

UART - Universal Asynchronous Receiver Transmitter

USB - Universal Serial Bus

VGA - Video Graphics Array

VHDL - VHSIC Hardware Description Language

VHSIC - Very High Speed Integrated Circuits

VLIW - Very Long Instruction Word

WCET - Worst Case Execution Time

XMD - Xilinx Microprocessor Debugger

XPS - Xilinx Platform Studio

Bibliografia

[AGP03] Luís Almeida, Bruno Gravato, and Bruno Pereira. ReTMiK - Real-Time Micromouse Kernel. http://sweet.ua.pt/∼lda/retmik/retmik.html, 2.edition, 2003.

[AP07] Luís Almeida and Paulo Pedreiras. Sistemas de Tempo-real Web Page. DETI-UA, http://sweet.ua.pt/∼lda/str/str.htm, September 2007.

[Arn07] Arnaldo Silva Rodrigues de Oliveira. *Especialização e Síntese de Processadores para Aplicação em Sistemas de Tempo-real*. PhD thesis, Universidade de Aveiro, 2007.

[AVI06] AVIX. AVIX-RT Web Page. http://www.avix-rt.com/html/specification.html, April 2006.

[Avn05] Avnet Electronics Marketing. Xilinx Spartan-3 Development Kit., January 2005. http://www.em.avnet.com/ads.

[Bak91] T. P. Baker. Stack-based scheduling of realtime processes. *Journal of Real-Time Systems*, 3(1):67–99, March 1991.

[But97] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Ed. Kluwer Academics Publishers, 1997.

[But05] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, volume 23 of *Real-Time Systems Series*. Springer, 2nd ed edition, 2005.

[Cel08] Celoxica. Celoxica RC10 Board, Web Page. http://www.celoxica.com/index.html, March 2008.

[Dig08] Digilent, Inc. Digital Design Engeneer's Source, April 2008. http://www.digilentinc.com/index.cfm.

[EEM08] EEMBC. The Embedded Microprocessor Benchmark Consortium, Web Page. http://www.eembc.org/, March 2008.

[FOF00] Joni da Silva Fraga, Rômulo Silva de Oliveira, and Jean-Marie Farines. *Sistemas de Tempo Real*. Florianópolis, July 2000.

[Fre08] FreeRTOS. The FreeRTOS.org Project Web Page. http://www.freertos.org/, May 2008.

[IBM08a] IBM. Developing PowerPC Embedded Application Binary Interface, Application Note, Version 1.0, April 2008. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF77852569970071B0D6/$file/eabi_app.pdf.

[IBM08b] IBM. IBM POWER Architecture Fact Sheet, April 2008. http://www-03.ibm.com/press/us/en/attachment/21546.wss?fileId=ATTACH_FILE3&fileName=POWER%20Timeline.pdf.

[IBM08c] IBM. Power Architecture licensing, April 2008. http://www-03.ibm.com/technology/ges/semiconductor/power/licensing/cores/ppc405.html.

[IBM08d] IBM. PowerPC 405 CPU Core Product Overview, April 2008. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/3D7489A3704570C0872571DD0065934E/$file/PPC405_Product_Overview_20060902.pdf.

[INC08] JMI SOFTWARE SYSTEMS INC. C EXECUTIVE and PSX REAL-TIME KERNELS Web Page. http://www.jmi.com/cexec.html, January 2008.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[Log08] Express Logic. Real-Time Operating Systems for Embedded Development, Web Page. http://www.rtos.com/, 2008.

[Lui90] Lui Sha, R. Rajkumar, J. P. Lehoczky. Priority Inheritance Protocols: An approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[LW82] J. Y. T. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. Performance Evaluation. 2(4):237–250, December 1982.

[Mic92] Micrium. Empowering embedded systems, Web Page. http://www.micrium.com/, 1992.

[Mic07] Microsoft Corporation. Windows Embedded CE. http://www.microsoft.com/windows/embedded/products/windowsce/default.mspx, July 2007.

[Mic08] SEGGER Microcontroller. Embeded Software Solutions, embOS Web Page. http://www.segger.com/embos_general.html, May 2008.

[MIP04] MIPS Technologies, Inc. MIPS SDE 5.03 Programmers Guide., January 2004. Document Number: MD00310, Revision 1.67.

[MRG01] Dan Ernst Todd M. Austin Trevor Mudge Richard B. Brown Matthew R. Guthaus, Jeffrey S. Ringenberg. Mibench: A free, commercially representative embedded benchmark suite. *IEEE 4th Annual Workshop on Workload Characterization*, December 2001.

[Pre07] Prevas. Sierra RTOS. http://www.prevas.com/product_development_rtos.html, July 2007.

[Rob08] Robert Bosch GmbH. CAN specification version 2.0., April 2008. http://www.can.bosch.com.

[Sha04] Sha, L. Abdelzaher, T. arzen, K. E. Cervin, A. Baker, T. Burns, A. Buttazzo, G. Caccamo, M. Lehoczky, J. Mok, A. K. Real time scheduling theory: A historical perspective. *Real Time Systems -AVENEI NJ-*, 28(2/3):101–155, 2004.

[Sof08] Green Hills Software. velOSity Real-Time Microkernel Web Page. http://www.ghs.com/products/micro_velosity.html, May 2008.

[Sol08] Eddy Solutions. Nimble Web Page. http://www.eddysolutions.com/, May 2008.

[SR04] Shift-Right. eXtreme Minimal Kernel Web Page. http://www.shift-right.com/xmk.htm, May 2004.

[Sys08] QNX Software Systems. QNX Realtime operating system Web Page. http://www.qnx.com/, May 2008.

[Win07] Wind River. VxWorks Technology, Web Page. http://www.windriver.com/vxworks/technology.html, July 2007.

[XES08] XESS. XSA-3S1000 Board, Web Page. http://www.xess.com/prod035.php3, March 2008.

[Xil08a] Xilinx, Inc. CPU Resets for Embedded Systems in FPGAs, February 2008. Xilinx GUIDANCE NOTE Ref # 077.

[Xil08b] Xilinx, Inc. EDK Concepts, Tools, and Techniques, April 2008. A Hands-On Guide to Effective Embedded System Design, EDK 9.2i User Guide.

[Xil08c] Xilinx, Inc. Fast Simplex Link, April 2008. Xilinx DS449 FSL Bus v2.11a specification, Data Sheet.

[Xil08d] Xilinx, Inc. FPGA, May 2008. http://www.xilinx.com/products/silicon_solutions/fpgas/.

[Xil08e] Xilinx, Inc. MicroBlaze Processor, March 2008. Xilinx UG081 (v9.0) MicroBlaze Processor, Reference Guide.

[Xil08f] Xilinx, Inc. MicroBlaze Processor Performance, April 2008. http://www.xilinx.com/products/design_resources/proc_central/microblaze_per.htm.

[Xil08g] Xilinx, Inc. On-Chip Peripheral Bus, March 2008. Xilinx DS401 OPB v2.0 specification, Data Sheet.

[Xil08h] Xilinx, Inc. Platform Studio and the EDK, March 2008. http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm.

[Xil08i] Xilinx, Inc. PowerPC 405 Processor, April 2008. Xilinx UG011 PowerPC Processor, Reference Guide.

[Xil08j] Xilinx, Inc. PowerPC 405 Processor, April 2008. Xilinx UG018 PowerPC Processor, Block Reference Guide.

[Xil08k] Xilinx, Inc. Processor Local Bus, April 2008. Xilinx DS531 PLB v4.6 specification, Data Sheet.

[Xil08l] Xilinx, Inc. Spartan-3 FPGA Family: Complete Data Sheet., May 2008. http://www.xilinx.com.

[Xil08m] Xilinx, Inc. Xilinx ISE Foundation, April 2008. http://www.xilinx.com/ise/logic_design_prod/foundation.htm.

[Xil08n] Xilinx, Inc. Xilinx Web Page., May 2008. http://www.xilinx.com.